Investigation numérique & terminaux Apple iOS : Acquisition de données

Mathieu Renard
mathieu.renard@ssi.gouv.fr

ANSSI

Résumé De nos jours l'investigation numérique se généralise à l'analyse de tout équipement informatique, incluant les ordiphones (*smartphones*) de toutes marques. À l'opposé des systèmes ouverts comme Android, il n'existe ni outil, ni méthodologie publique permettant d'acquérir le contenu du système de fichiers d'un terminal Apple récent. Or, cette étape est incontournable lors de la recherche de preuve informatique visant à infirmer ou confirmer une compromission. Afin de répondre à cette problématique, ce document commence par décrire succinctement l'architecture et les mécanismes de sécurité du système iOS d'Apple avant de revenir sur les méthodes d'acquisition de données existantes et d'en présenter les limites. Enfin, cet article présente deux techniques d'extraction logique dont une résultant de l'analyse et de la mise en œuvre des méthodes initialement utilisées par les logiciels de débridage, plus connus sous le nom de *jailbreak*.

1 Introduction

L'usage de terminaux mobiles (*smartphones*, tablettes) s'est démocratisé dans le monde professionnel, impliquant le stockage et le traitement d'informations professionnelles sur des équipements mal maîtrisés. Ces équipements deviennent donc, peu à peu, des cibles de choix pour les attaquants. Par conséquent, il est nécessaire de définir les méthodes et les outils de recherche de preuves appropriés afin de pouvoir identifier et fournir tous les éléments matériels permettant d'infirmer ou de confirmer une compromission.

Apple, qui souhaite maîtriser ses plateformes mobiles, garde l'accès au code source des composants du système iOS privé. Seul le code source de quelques composants est accessible publiquement [22]. Les plateformes mobiles d'Apple étant fermées, la liste d'outils d'investigation numérique disponible publiquement se limite principalement au framework iPhone-DataProtection [15] et à l'outil iPhoneBackupAnalyser2 [36]. Les outils du framework iPhoneDataProtection, sont principalement basés sur une vulnérabilité, affectant la chaîne de démarrage. Apple ayant corrigé cette

vulnérabilité, ces outils sont en partie inutilisables sur les terminaux récents. L'usage de l'outil iPhoneBackupAnalyser2, qui est basé sur l'acquisition des données stockées dans les sauvegardes du terminal ne convient pas à l'analyse d'un terminal potentiellement compromis. Il est donc nécessaire de définir une nouvelle approche permettant l'extraction et l'analyse de données de terminaux récents.

Cette étude suppose que l'expert réalise l'analyse du terminal à la demande de son propriétaire (il dispose donc des éléments autorisant le déverrouillage du terminal). Dans ce contexte, cet article vise à établir un état des lieux des techniques d'investigation numérique sur terminaux iOS. Il commence par une brève description de l'architecture et des mécanismes de sécurité du système iOS, avant de revenir sur les méthodes d'acquisition physique de données et d'en présenter les limites.

Deux techniques d'acquisition logiques des données seront présentées afin de répondre aux problématiques identifiées. La première méthode proposée est dite « sans altération ». Celle-ci s'appuie sur l'utilisation du protocole d'échange entre le système iOS et l'outil de gestion iTunes d'Apple. Cette technique étant limitée par les fonctionnalités de sécurité du système, une seconde méthode d'acquisition dite « avec altération » est développée. Celle-ci s'appuie sur l'étude des logiciels de débridage, afin de contourner les limites identifiées précédemment.

2 Architecture matérielle d'un terminal iPhone 5S

L'architecture matérielle de la plateforme iOS représentée par la figure 1. Cette architecture repose principalement sur le SoC (*System On Chip*) Apple A7. Ce composant héberge notamment un processeur bicœur ARM 64, ainsi qu'un crypto-processeur utilisé pour les opérations de chiffrement et déchiffrement des données stockées sur la mémoire flash de type NAND. Le crypto-processeur étant placé en coupure entre la mémoire RAM et la NAND, toutes les données provenant de la mémoire et, à destination de la NAND sont chiffrées. L'analyse des composants présents sur la carte mère de l'iPhone 5S présentée sur la figure 2 met également en évidence divers composants radio dont le Modem 4G/LTE de Qualcomm qui est en charge des différents modes de communication radio. Les composants embarqués sur la carte mère se présentent sous la forme de boîtiers à matrice de billes (BGA). Un autre élément important du point de vue de la sécurité matérielle est l'inaccessibilité des bus de données. Il n'est pas possible d'acquérir les données échangées entre le SoC et la NAND à l'aide de sondes matérielles et d'un analyseur logique.

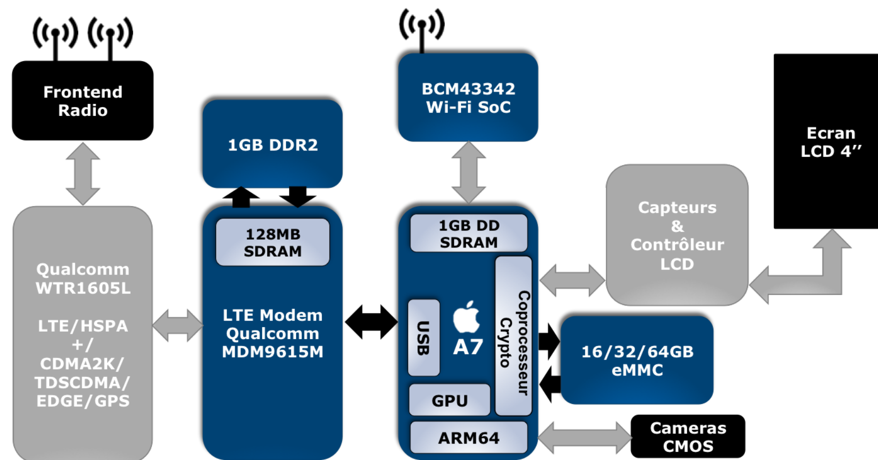

Figure 1. Architecture matérielle de la plateforme iOS

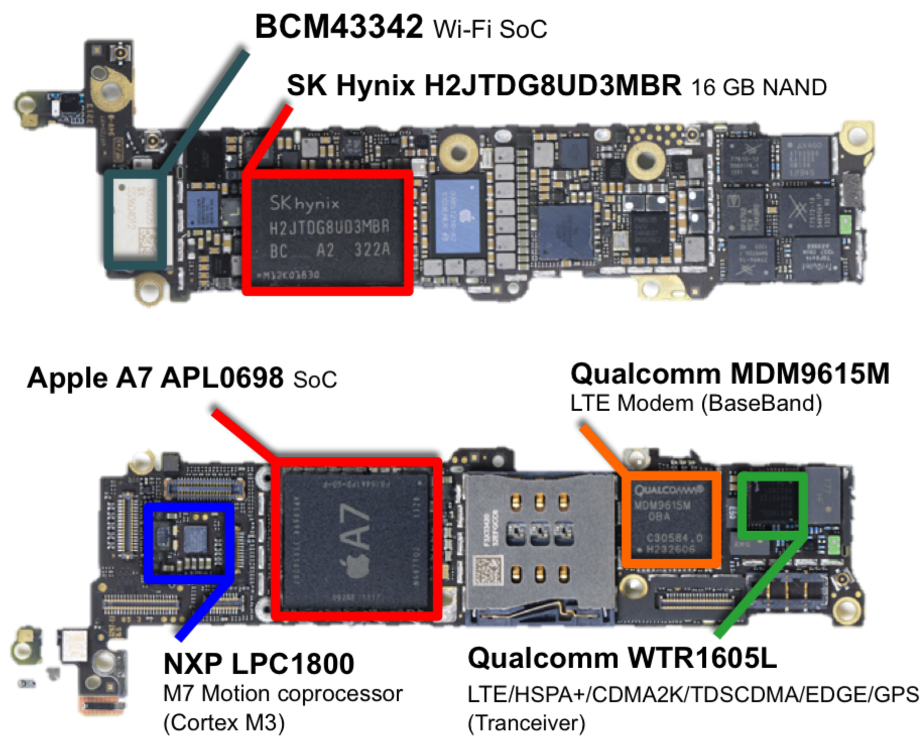

Figure 2. Vue interne d'un iPhone 5S (chipworks [41])

3 Architecture logicielle d'un terminal iOS

La figure 3 présente l'architecture logicielle d'un terminal iOS. Ce système est organisé en différents niveaux d'abstraction dont l'accès est géré par les mécanismes de sécurité implémentés dans le noyau.

3.1 XNU

Le noyau XNU est le cœur du système iOS. Il s'agit d'un noyau hybride qui intègre à la fois les fonctionnalités d'un noyau monolithique et celles d'un micro noyau (figure 4).

XNU est composé de 3 composants majeurs :

- Le micro-noyau Mach est en charge de la gestion du processeur, de la planification des tâches et de la gestion des communications entre les processus (IPC) ;
- La couche BSD intègre plusieurs fonctionnalités du noyau FreeBSD. XNU hérite des piles protocolaires du noyau FreeBSD, des fonctionnalités de gestion des permissions et des utilisateurs, du système de contrôle d'accès obligatoire (MAC) ;
- I/O Kit gère les interfaces avec les différents périphériques.

Lorsque le système est démarré, XNU contrôle l'intégrité du système [33,34].

3.2 Couche de service et interface utilisateurs

La couche *Core Services* fournit les bibliothèques de fonctions de base aux applications. Ces bibliothèques offrent une couche d'abstraction des fonctionnalités de sécurité, de gestion du contenu multimédia, de stockage distant (iCloud) et d'affichage. Celles-ci sont organisées en deux familles de bibliothèques. L'accès aux bibliothèques privées, permettant d'accéder aux fonctionnalités critique du système et aux données personnelles de l'utilisateur, est restreint aux applications d'Apple.

3.3 Applications iOS

Il existe deux familles d'applications sur iOS : les applications du système et les applications tierces. Ces dernières ne sont pas autorisées à utiliser les fonctionnalités offertes par les bibliothèques privées. Apple s'assure que cette politique est respectée en obligeant contractuellement et fonctionnellement les développeurs à faire valider leurs applications avant d'autoriser leur publication sur l'App Store. Techniquement, ces restrictions se traduisent par la signature obligatoire des applications tierces par une autorité issue de l'IGC d'Apple avant publication.

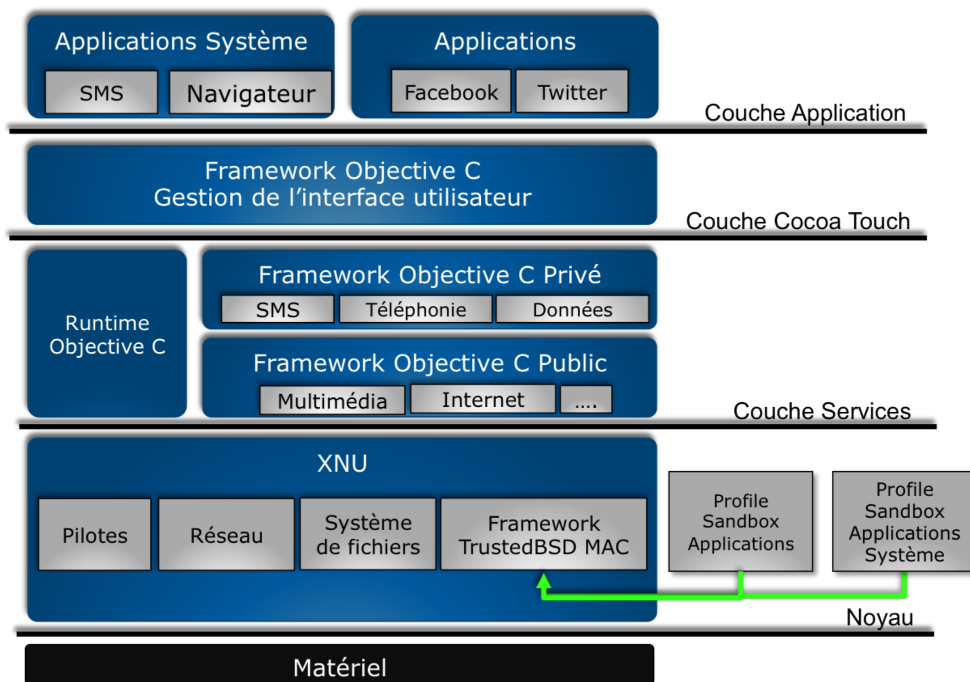

Figure 3. Organisation du système iOS

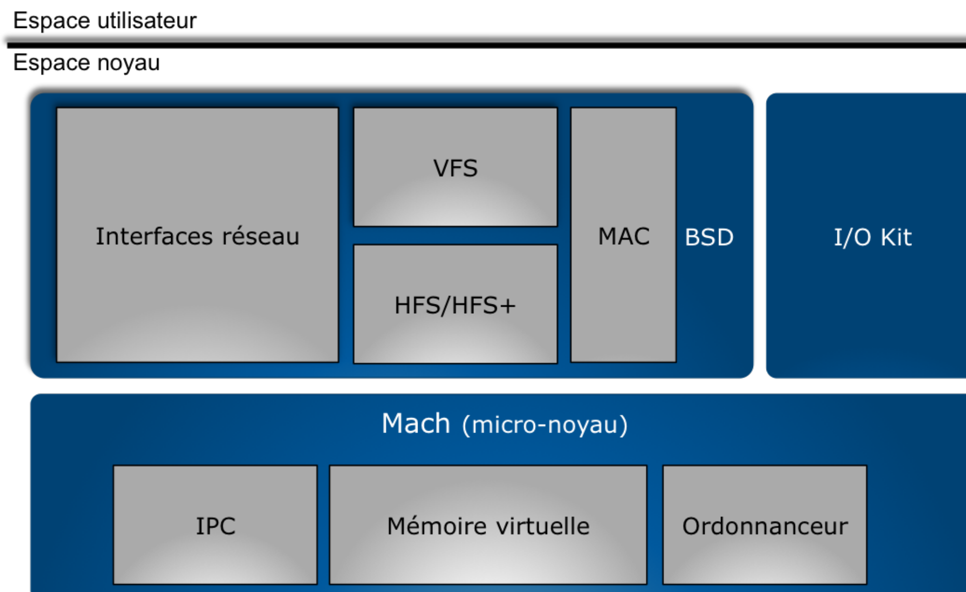

Figure 4. Architecture du noyau XNU

4 Surface d'attaque et mécanismes de sécurité d'iOS

Apple s'attache à réduire la surface d'attaque de ses équipements à chaque nouvelle version matérielle et logicielle. Ainsi, un grand nombre de mécanismes de sécurité a été intégré au système. Les plus importants sont présentés dans les chapitres suivants.

4.1 Intégrité du système lors du démarrage

Apple, qui dispose d'une totale maîtrise de la couche matérielle, a sécurisé son système de bout en bout. L'intégrité du système au démarrage est garantie par l'implantation d'une racine de confiance dans la *BootROM* de l'équipement. Celle-ci est utilisée pour garantir l'intégrité du système de démarrage et des mises à jours du système.

Démarrage en mode nominal Les étapes de démarrage en mode nominal sont présentées sur la figure 5.

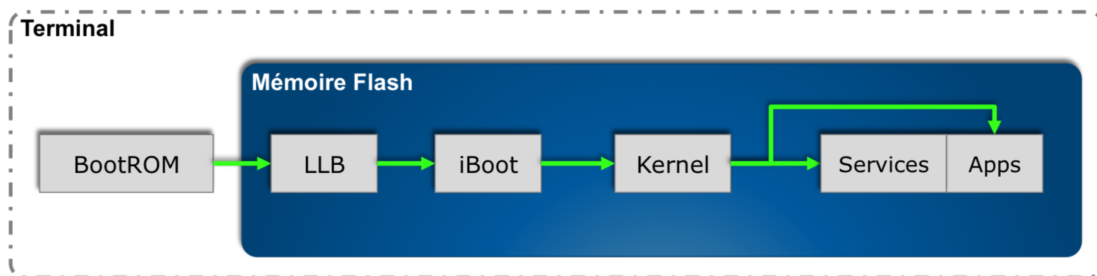

Figure 5. Chaîne de confiance du système iOS

BootROM & Racine de confiance : Le premier code exécuté par le processeur est un code immuable gravé dans le SoC. Ce code est chargé de l'initialisation des périphériques de premier niveau tel que la NAND, puis de vérifier l'intégrité du code du chargeur de démarrage de bas niveau. Le bootROM est gravé sur la puce de silicium, par conséquent il n'est pas possible de le mettre à jour ou de le modifier. Une vulnérabilité affectant ce composant ne peut pas être corrigée par Apple sur les terminaux existants.

Chargeur de démarage de bas niveau (LLB) : Une fois chargé et exécuté, le chargeur de démarrage de bas niveau (LLB – *Low Level Bootloader*) poursuit l'initialisation des périphériques, vérifie l'intégrité du chargeur de démarrage de niveau supérieur : *Boot* et lui passe la main.

iBoot : *iBoot*[1] est le chargeur de démarrage de haut niveau. Il termine l'initialisation des différents périphériques et vérifie l'intégrité du noyau avant de lui passer la main.

Démarrage en Mode DFU ou Mode de récupération Ces modes de démarrage alternatifs peuvent être exécutés soit à la demande de l'utilisateur (par exemple lors du déploiement d'une mise à jour du système ou lors de la réinitialisation de l'équipement), soit lorsque le système détecte une anomalie. Le fonctionnement du mode DFU (*Device Firmware Update*) et du mode de récupération sont très similaires. Les étapes de démarrage en mode DFU sont présentées sur la figure 6. Lors du démarrage en mode DFU, le code du BootROM attend qu'iTunes lui envoie une version allégée d'*iBoot* dénommée *iBSS* (Chargeur de démarrage de second niveau). Le *BootROM* procède alors à la vérification de la signature de l'*iBSS* avant de passer la main.

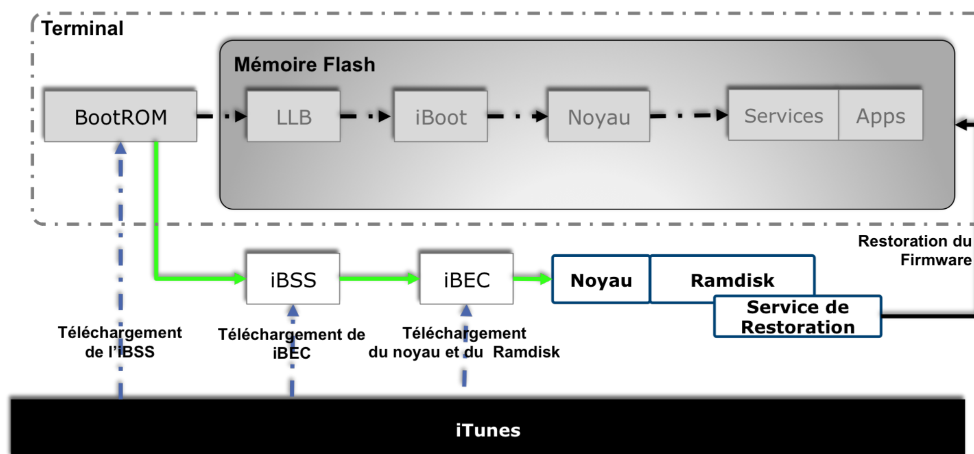

FIGURE 6. Chaîne de confiance (mode DFU)

iBSS, iBEC & Ramdisk : L'*iBSS* permet de vérifier la signature de l'*iBEC* (Chargeur de démarrage de troisième niveau), qui à son tour vérifie la signature du *Ramdisk* de restauration avant chargement et exécution. L'*iBSS* est également en charge des fonctionnalités de protection anti-rejeu des mises à jour et d'anti-retour-arrière de la version d'iOS. Les fonctions de vérification de l'intégrité et de l'authenticité de l'image du *firmware*

1. iBoot ayant fait l'objet d'une étude approfondie de la part des membres du projet *OpeniBooT* [19]. Il est recommandé de consulter le site du projet pour obtenir des informations détaillées concernant cet élément.

à réinstaller sont gérées par le binaire `/usr/sbin/asr` contenu dans le disque de restauration.

4.2 Intégrité du système en fonctionnement

L'intégrité du système repose sur plusieurs mécanismes de sécurité :

- Restriction des privilèges : Apple a conçu la plate-forme iPhone avec l'intention d'appliquer la politique de moindre privilège. Par conception les applications tierces sont exécutées avec les droits de l'utilisateur `mobile` qui dispose de droits restreints.
- Signature de code et contrôle d'intégrité des binaires : Apple a intégré un mécanisme de signature des exécutables. L'objectif de ce mécanisme est de garantir que seules les applications approuvées non modifiées sont exécutées sur l'appareil.

Il existe quatre modèles de distribution d'application :

- Développement sur l'appareil : permet aux développeurs de créer et tester des applications sur leurs propres dispositifs ;
- Distribution *Ad-Hoc* : permet aux développeurs d'applications de réaliser des tests sur une centaine d'équipements pré-déclarés ;
- Distribution Locale : permet aux développeurs d'entreprises de distribuer leurs applications sur n'importe quel terminal ;
- Distribution sur *l'App Store* : Permet aux développeurs de publier des applications sur *l'iTunes App Store*. Toutes les applications de *l'App Store* sont examinées par Apple avant publication afin de vérifier leur compatibilité avec la charte de développement d'Apple.

Le mode d'adhésion au programme de développement d'Apple détermine le mode de distribution autorisé.

4.3 Isolation des applications

Toutes les applications tierces et certaines applications systèmes sont cloisonnées. Cette politique de cloisonnement s'appuie sur un ensemble de règles appliquées par les mécanismes de contrôle d'accès du framework Mandatory Access Control (MAC) de TrustBSD :

- Définition d'un répertoire de base unique par application ;
- Restriction d'accès aux données d'autres applications ;
- Restriction d'accès aux services et applications du système ;
- Restriction du temps d'exécution en arrière plan ;
- Définition de groupes d'accès au trousseau de clés partagé ;
- ...

L'accès par des applications tierces aux données de l'utilisateur et l'utilisation des fonctionnalités telles que iCloud sont contrôlés à l'aide de droits spécifiques appelés *entitlements*. Ce même mécanisme est utilisé pour autoriser l'application *Mobile Safari* à exécuter du code dynamique non signé utilisant la compilation à la volée, (JIT - *just-in-time*), afin d'accélérer l'exécution de code JavaScript. Le tableau 1 présente un exemple de droits spécifiques.

Description	Entitlement
Accès aux documents stockés dans le cloud	`com.apple.developer.ubiquity-container-identifiers`
Stockage de document dans le cloud	`com.apple.developer.ubiquity-kvstore-identifier`
Accès aux URL sensibles (Configuration...)	`com.apple.springboard.opensensitiveurl`
Exécution du wipe	`com.apple.springboard.wipedevice`
Signature dynamique de code (JIT)	`dynamic-codesigning`
Autorisation de debogage par une autre application	`get-task-allow`
Autorisation de debogage d'une autre application	`com.apple.springboard.debugapplications`
Lancement d'application	`launchcom.apple.springboard.launchapplications`

Table 1. Exemple de droits spécifiques

4.4 Restriction de l'exécution de code malveillant

Afin de complexifier la tâche des attaquants, des mécanismes de sécurité sont intégrés au système iOS [2]. La liste suivante résume la liste des mécanismes de sécurité les plus importants :

- Activation du bit *Never eXecute* (NX) supporté par le processeur ARM, intégré au SoC ;
- *Address Space Layout Randomization* (ASLR), pour les espaces d'adressage utilisateur et noyau ;
- *Position-independendent executable* (PIE), activé par défaut sur les chaînes de compilation d'applications Apple ;
- *Stack & Heap canaries* ;
- ...

2. Pour plus d'information concernant ces mécanismes de protection le lecteur est invité à consulter les travaux de Stefan Esser [11,10,12], Charlie Miller [33,34] Mark Dowd et Tarjei Mandt [9,32] qui détaillent précisément l'implémentation de ces fonctionnalités.

4.5 iPhone data protection

Apple a également intégré des fonctionnalités de sécurité dans le but de garantir la confidentialité des données stockées sur l'équipement [3].

Protection des données utilisateurs Les données stockées sur la NAND sont chiffrées avec une clé de volume. Cette clé de volume est chiffrée à l'aide de la clé *0x89B* qui n'existe qu'en mémoire noyau et provient de la dérivation de la clé UID.

$$K_{0x89B} = \texttt{AES_encrypt}(UID, 183E99676BB03C546FA468F51C0CBD49)$$

Cette clé, unique et propre à chaque terminal, est enfouie dans le processeur cryptographique. Cette implémentation garantit la confidentialité des données lorsque l'équipement est éteint et autorise l'effacement d'urgence.

En plus de cette couche de chiffrement intégral, Apple a mis en place différentes classes de protections permettant de restreindre l'accès aux fichiers de l'utilisateur et du système. La figure 7 présente les classes de protections utilisées pour les données.

Accessibilité	**Fichiers** NSFileProtection...	**Keychain** kSecAttrAccessible...
Déverrouillé...	**Complete**	**... WhenUnlocked && ThisDeviceOnly**
Verrouillé...	CompleteUnlessOpen	N/A
Après le premier déverrouillage...	CompleteUntil FirstUserAuthentication...	AfterFirstUnlock && ThisDeviceOnly
Toujours	... None	... Always && ThisDeviceOnly

Figure 7. Classes de protections

En résumé, chaque fichier possède une clé de chiffrement unique utilisée pour chiffrer son contenu. Cette clé dénommée *File key*, est protégée par la clé maitre de la classe de protection choisie (AES-wrap - RFC 3394). La *File Key* est stockée dans les *métadatas* du fichier qu'elle protège.

Les fichiers protégés avec la classe *NSProtectionNone* sont chiffrés (AES-256) avec une clé maitre stockée dans la zone effaçable de la NAND. Cette clé est elle-même chiffrée (AES-wrap) par la clé *0x835* qui existe uniquement dans la mémoire noyau et est issue de la dérivation de la clé UID et d'une constante.

$$K_{0x835} = \texttt{AES_encrypt}(UID, 01010101010101010101010101010101)$$

3. Les équipes de Sogeti ESEC [15] et d'Elcomsoft [12] ayant abondamment documenté ces éléments, le lecteur intéressé peut de se reporter à leurs travaux respectifs pour obtenir plus de détails sur l'implémentation de ces fonctionnalités.

Les autres clés maîtres sont stockées dans le fichier *systembag.kb*, qui est protégé avec la classe de protection *NSProtectionNone*. Le contenu de ce fichier est sur-chiffré à la fois par la *Password Key* et la clé *Bag1* qui est stockée en clair dans la mémoire effaçable. La *Password Key* est le résultat de la dérivation de la clé UID et du mot de passe de l'utilisateur (Algorithme de dérivation Tangling [23,6,40]).

Le figure 8 résume les différentes opérations qui interviennent dans la protection des données de l'utilisateur.

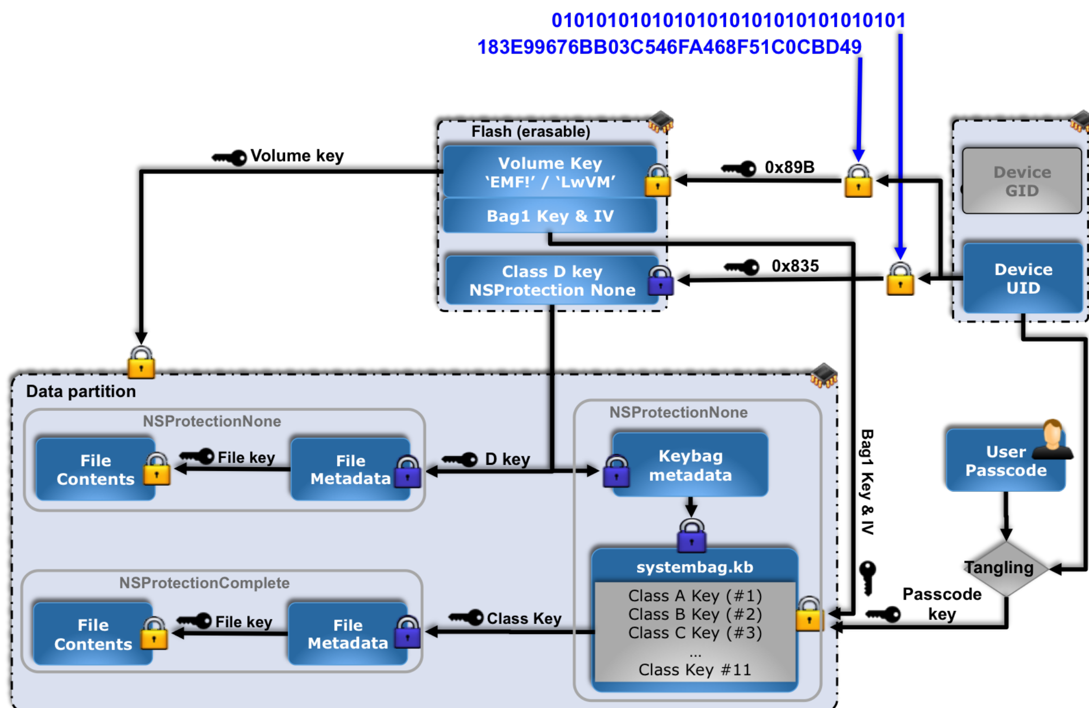

Figure 8. iPhoneDataProtection

Effacement d'urgence Lorsque l'effacement d'urgence est déclenché, les clés stockées dans la zone effaçable de la NAND sont détruites, rendant l'accès aux données stockées impossible.

4.6 Conclusion sur les mécanisme de sécurité

Apple fait beaucoup d'efforts pour garantir l'intégrité du système iOS et protéger les données stockées en cas de vol du terminal. Toutefois, pour chaque version d'iOS, il existe des vulnérabilités majeures qui, lorsqu'elles sont associées, permettent de contourner tout ou partie des mécanismes de

sécurité. Il est donc nécessaire de pouvoir disposer d'outils et de méthodes d'acquisition de données afin de permettre l'analyse de ces terminaux et d'infirmer ou confirmer la compromission de l'équipement.

4.7 Extraction physique de données

Comme la plupart des systèmes embarqués, les partitions de données sont stockées sur la NAND du terminal. Sur les terminaux Apple la taille de cette mémoire non volatile peut varier de 4 à 128Go. L'acquisition physique de données réalisée par le démontage de la NAND est la méthode la plus fiable pour extraire les données d'un système compromis. Toutefois, les données stockées sur la NAND d'un terminal Apple sont chiffrées. L'analyse des données brutes est alors inutile sans connaissance des clés enfouies dans le crypto-processeur [5] [4]. En 2011, les chercheurs de Sogeti ESEC ont présenté une méthode d'acquisition de données à partir d'un système alternatif [15]. Cette méthode d'acquisition de bas niveau s'appuie sur une vulnérabilité dans un des composants de la chaîne de démarrage pour extraire le contenu de la NAND.

Mode DFU & Limera1n Lors de la mise à jour et de la réinitialisation du système, iTunes utilise le protocole DFU. Ce protocole est défini dans les standards USB [1]. L'analyse des échanges entre iTunes et un terminal, permet d'identifier les commandes présentées sur le listing 1 :

```
/*
usb_control_msg -- Send a control message to a device
int usb_control_msg( usb_dev_handle *dev, int requesttype,
                     int request, int value, int index,
                     char *bytes, int size, int timeout);
*/

//get status
usb_control_msg(idev, 0xA1, 3, 0, 0, buf, 6, 1000);

//send command
usb_control_msg(idev, 0x40, 0, 0, 0, buf, strlen(buf), 1000);

//send file
usb_control_msg(idev, 0x21, 1, fileid, 0, paketid, s, 1000);
...
```

Listing 1. Exemple de commandes DFU Apple [25]

4. A ce jour, il n'existe aucune étude publique concernant l'extraction de la clé enfouie par canaux cachés. Il faut donc avoir recours à d'autres méthodes d'acquisition.

Publié en 2010 par George Hotzk *alias* GeoHot, *Limera1n* [18], permet de démarrer un système alternatif personnalisé. En effet, cet outil tire partie d'une erreur de programmation dans la fonction de gestion de message : `usb_control_msg(0x21, 2)` intégrée au code du *BootROM* des terminaux équipés du processeur Apple A4 ou inférieur (iPhone 3/3GS/4, iPod Touch 3G/4G) [5] pour désactiver les fonctions de contrôle de signature.

Lecture de la NAND à partir d'un système Alternatif Les outils *redsnow* [24] et *opensnow* [42] utilisent *Limera1n* pour démarrer un système personalisé de type *RamDisk*. Le framework iPhoneDataProtection [7] permet de créer un système alternatif [39] contenant les outils nécessaires à l'acquisition des données stockées sur la NAND du terminal (copie physique des partitions système et données utilisateurs). Les différentes étapes de démarrage à partir d'un *RamDisk* sont présentées sur la figure 9.

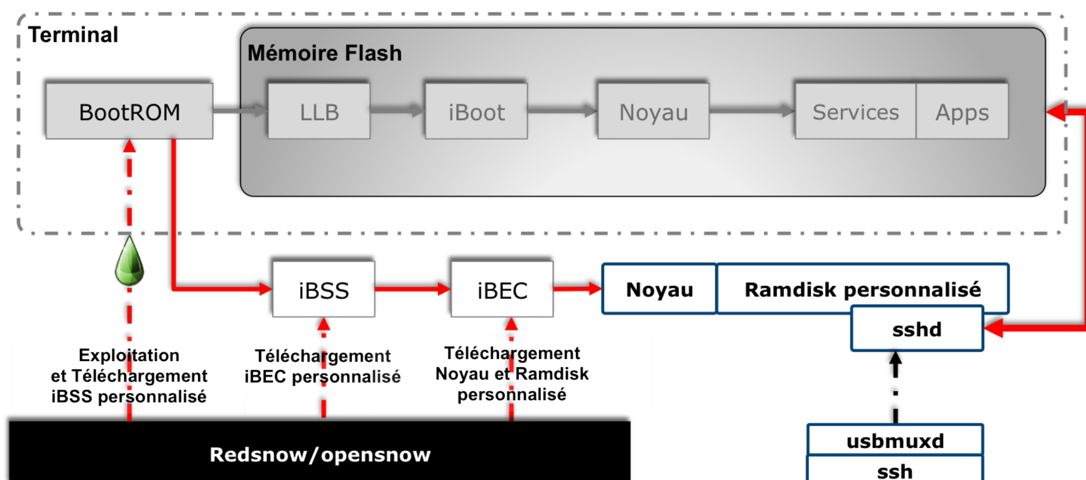

Figure 9. Aquisition physique de donnée

Une fois le système alternatif démarré, il est possible de procéder à l'acquisition des données et d'utiliser le crypto-processeur afin de calculer les clés *0x89B* et *0x835*. Il devient possible de déchiffrer et d'analyser les données. Cette technique stable est implémentée dans diverses solutions industrielles d'acquisition et d'analyse de données.

Flash Translation Layer (FTL) & Récupération de données Comme un disque dur, une mémoire flash de type NAND est organisée en

5. Les terminaux équipés de processeurs Apple A5 et supérieurs (iPhone 4S, iPad 2...) ne sont pas vulnérable à cette attaque.

secteurs. L'effacement est géré par bloc de données, ceci afin de réduire le nombre d'opérations d'effacement et de préserver la durée de vie de la mémoire flash.

Du point de vue du système, la mémoire flash est vue comme un périphérique de type bloc, grâce à la *Flash Translation Layer* (FTL). Cette couche d'abstraction a été développée pour optimiser les performances et la durée de vie des secteurs de la NAND. Sur les terminaux iOS, la FTL est intégrée dans le code du *boot loader* et dans le noyau XNU. Le projet open source *OpeniBoot* [19], issu de l'analyse par rétro ingénierie de la chaîne de démarrage, embarque une version libre de la FTL utilisée par iOS. Cette fonctionnalité intégrée au framework iPhoneDataProtection offre dans certains cas, la possibilité d'extraire les données effacées des systèmes iOS affectés par une vulnérabilité dans la chaîne de démarrage.

Conclusion L'extraction physique des données d'un iPhone est réalisable uniquement en s'appuyant sur une vulnérabilité dans la chaîne de démarrage du terminal. Il est alors possible d'extraire l'intégralité du système de fichiers ainsi que le contenu de certains fichiers effacés ou modifiés par l'utilisateur.

Cependant, à la date de rédaction de ce document, il n'existe aucune vulnérabilité publique affectant la *BootROM* des iPhones et iPad récents (iPhone 4s/5/5s, iPad 2 et supérieur). L'acquisition physique n'étant pas réalisable sur les terminaux récents, il convient d'utiliser une autre méthode pour extraire et analyser le contenu d'un équipement Apple.

5 Extraction logique de données (sans altération)

L'extraction logique implique l'acquisition des données utilisant le système d'exploitation de l'appareil à l'aide d'un ensemble connu de commandes. En mode nominal, les terminaux Apple exposent plusieurs configurations USB. Chacune de ces configuration USB permet d'accèder à un ou plusieurs services : *Picture Transfer Protocol* (PTP), *Apple Mobile Device* et *Apple USB Ethernet*. En mode de récupération, seul le service *Device Firmware Update* (DFU) est exposé. La figure 10 présente les configurations exposées par un iPhone 4 executant iOS 7.0.6. Lors d'une investigation numérique, l'utilisation des services PTP et *Apple Mobile Device* permet d'acquérir certaines données stockées sur le terminal.

Configuration	Fonctions suportées
PTP	Transfert de fichiers multimedias
iPod USB Interface	Contrôle des fonctionalitées multimedias
PTP + Apple Mobile Device	Transfert de fichiers multimedias & Services iTunes
PTP + Apple Mobile Device + Apple USB Ethernet	Transfert de fichiers multimedias, Services iTunes & Interface réseau virtuelle)

Figure 10. Configurations USB exposées par un iPhone 4 (iOS 7.0.6 - Mode nominal)

5.1 Picture Transfer Protocol (PTP)

Tous les terminaux Apple supportent le *Picture Transfer Protocol* (PTP). PTP est un protocole développé par l'International Imaging Industry Association pour permettre le transfert d'images depuis un appareil photo numérique sur un ordinateur (ISO 15740).

```
$ gphoto2

--summary

Camera summary:
Manufacturer: Apple Inc.
Model: Apple iPad
  Version: 7.0.4
  Serial Number: XXXXXXXXXXXX

Vendor Extension ID: 0x6 (1.0)
Vendor Extension Description: Device has no vendor extensions
PTP Standard Version: 1.10
Capture Formats: Association/Directory JPEG PNG TIFF
Display Formats:
    Undefined Type, Association/Directory,
    Script, AIFF,
    MS Wave, MP3,
    MS AVI, MPEG,
    ASF, Apple Quicktime,
    JPEG, PNG,
    TIFF

[..]

Device Capabilities:
    File Download, File Deletion,
    No File Upload, No Image Capture,
    No Open Capture, No vendor specific capture

[..]
```

Listing 2. Énumération des fonctionalités PTP supportées par un iPad

Ainsi, il est possible à l'aide d'un utilitaire tel que *gphoto2* de récupérer le contenu du répertoire `/var/mobile/Media/DCIM`. Le listing 3 présente les résultats obtenus lors de l'éxecution de la commande sur un iPad.

```
$ gphoto2 --list-files

There is no file in folder '/'.
There is no file in folder '/store_00010001'.
There is no file in folder '/store_00010001/DCIM'.
There is 1 file in folder '/store_00010001/DCIM/815ECHXL'.
#1     IMG_6806.JPG               rd  6454 KB 5184x3456 image/jpeg

There are 4 files in folder '/store_00010001/DCIM/860OKMZO'.
#2     IMG_0006.JPG               rd  1906 KB 2592x1936 image/jpeg
#3     IMG_0008.JPG               rd  1016 KB 1365x2048 image/jpeg
#4     IMG_0009.JPG               rd  1942 KB 2592x1936 image/jpeg
#5     IMG_0010.JPG               rd  1010 KB 1365x2048 image/jpeg

[..]
```

Listing 3. Enumération des photos stockées sur le terminal

5.2 Services iTunes

Outre ses fonctionnalités multimédia, iTunes est utilisé pour administrer les différents équipements Apple. À ce titre, iTunes est à même de démarrer et communiquer avec les différents services hébergés sur les terminaux. L'étude des échanges entre iTunes et un terminal fonctionnant en mode nominal, permet d'identifier les différents protocoles d'échanges.

USBmux iTunes communique avec l'iPhone en utilisant le protocole *USBmux* qui réalise le multiplexage de connexion TCP sur un *pipe* USB. Sur Mac OSX, le service *USBmuxd* est lancé par *launchd* au démarrage du système. Les applications tierces qui souhaitent échanger des données avec un terminal Apple doivent soit passer par le service *usbmuxd*, soit re-implémenter ce protocole propriétaire.

En 2009, Hector Martin alias « marcan » a développé une alternative open-source [4]. Il est ainsi possible de communiquer avec un terminal Apple depuis tous les systèmes supportant cette bibliothèque. Ce service crée une socket UNIX à l'emplacement `/var/run/usbmuxd` et se place en attente de connexion d'un client. Lorsqu'un terminal fonctionnant en mode nominal est connecté, *USBmuxd* ouvre la connexion, puis commence à relayer les commandes qu'il reçoit au travers de socket UNIX. Le listing 4 présente les structures de données à utiliser pour communiquer avec le service USBMuxd.

```
struct usbmux_header
{
    u32 length;    // Longueur totale du message
    u32 reserved;  // 0
    u32 type;      // Type de message
    u32 tag;    // Identifiant du message (retourne dans la reponse)
};

struct usbmux_result
{
    struct usbmux_header header;
    u32 result;
};

struct usbmux_connect_request
{
    struct usbmux_header header;
    u32  device_id;
    u16  port;
};

enum
{
    usbmux_result  = 1,
    usbmux_connect = 2,
    usbmux_hello   = 3,
};
```

Listing 4. Structures de données USBMux (*little-endian*) [26]

La figure 11 présente l'établissement d'une session entre un client et le service *USBmuxd*, en écoute sur le poste de travail de l'utilisateur. Une fois la session établie les données envoyées par le client sur le `/var/run/usbmuxd` sont relayées vers le service en écoute sur le terminal.

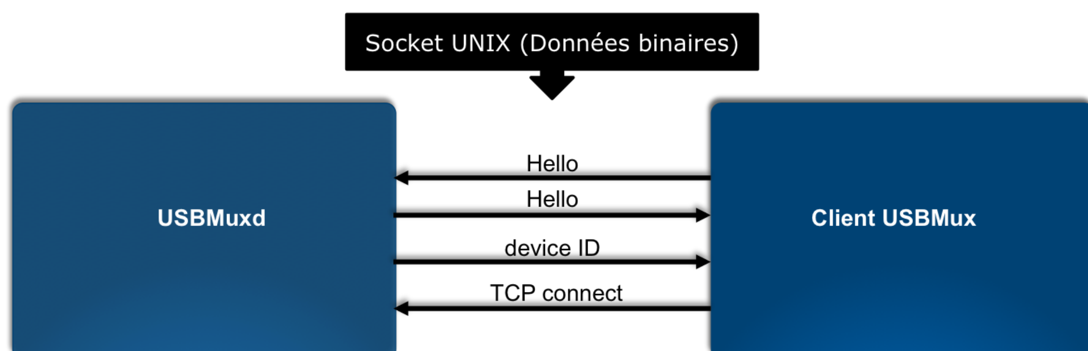

Figure 11. Communicating with usbmuxd

Lockdownd Le binaire lockdownd lancé au démarrage du terminal est responsable de plusieurs tâches de sécurité incluant l'activation du terminal et la délégation des communications à d'autres services. Ce service est en écoute sur le port 62078. Il communique en utilisant le protocole *USBmux*. La figure 12 présente les échanges réalisés lors de l'appairage d'un terminal avec un périphérique. Comme pour tous les services *iTunes*, les données échangées entre le client et le service sont échangées au format *plist*. Les échanges peuvent être chiffrés (utilisation de SSL) à la demande du terminal.

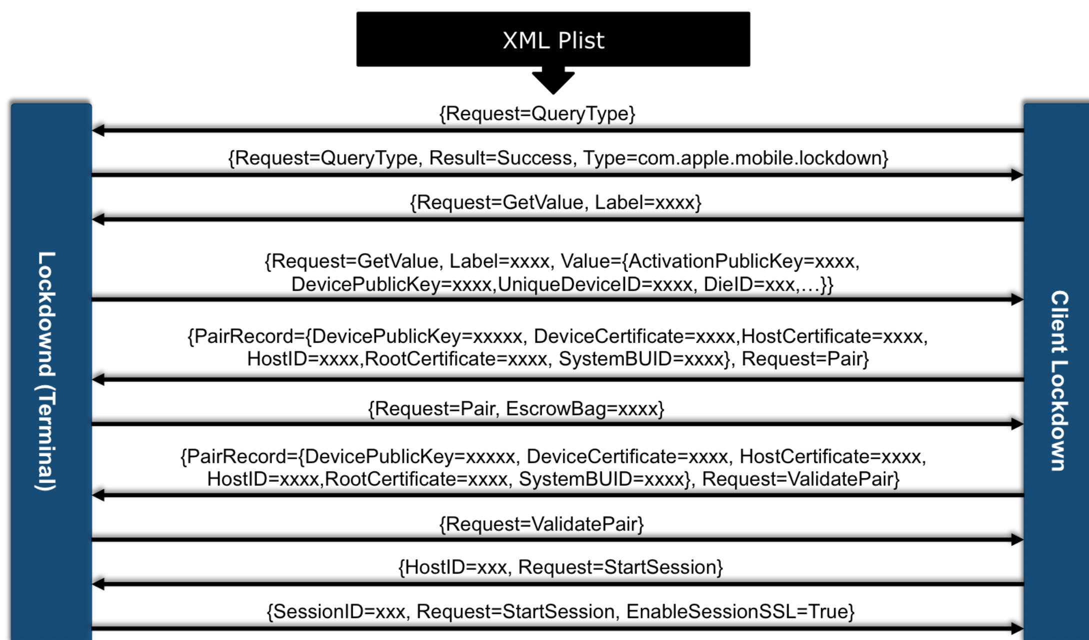

Figure 12. Protocol d'appairage

L'appairage n'est réalisable que lorsque le terminal est déverrouillé et requiert l'accord de l'utilisateur. Le message de validation présenté sur la figure 13.

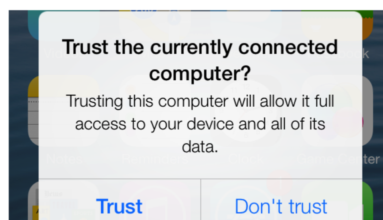

Figure 13. Demande de validation de l'appairage par l'utilisateur

Ce point n'est pas bloquant dans le cadre d'une investigation numérique visant à infirmer ou confirmer une compromission. En effet, ce type d'analyse est réalisé à la demande de l'utilisateur qui est en mesure de transmettre ou retirer le mot de passe d'accès à son terminal.

Bibliothèques MobileDevice et libImobiledevice Libimobiledevice [4] est une implémentation open-source et multiplateformes développée par Nikias Bassen. Tout comme son homologue propriétaire *MobileDevice*, cette bibliothèque fournie des interfaces permettant aux développeurs d'interagir avec les services iTunes. La figure 14 synthétise les différents échanges intervenants lors de l'établissement d'une connexion entre une application et un service iTunes.

Les chapitres suivants présentent les services *iTunes* intéressants du point de vue de l'analyste en recherche de preuve.

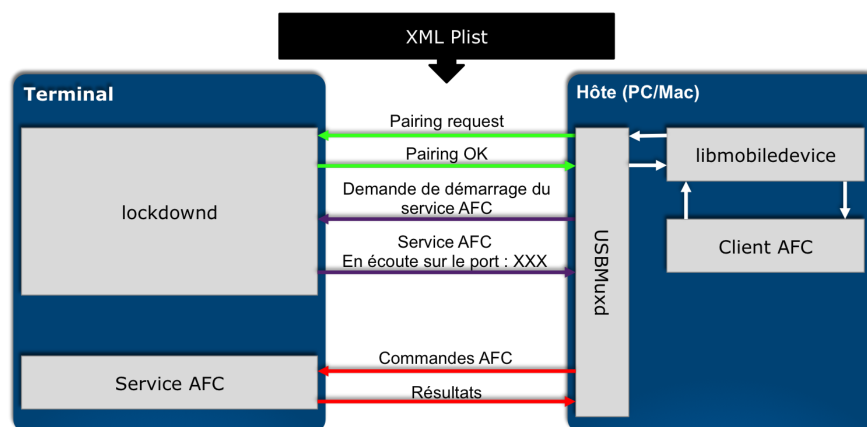

Figure 14. Échanges entre une application et un service iTunes

com.apple.afc : Le protocole Apple File Communication (AFC) est utilisé pour communiquer avec le service *afcd*. Ce service est utilisé par iTunes pour transférer les fichiers lors des procédures de sauvegarde, de restauration et de synchronisation. La politique de sécurité appliquée sur le service *afcd* lui interdit tout accès en dehors du répertoire : `/var/mobile/Media`.

com.apple.mobile.house_ arrest : Ce service permet d'accéder aux dossiers de base des applications tierces déployées sur le système. En d'autres termes, en utilisant un client AFC au travers du service *house_ arrest*, il est possible de télécharger les binaires, les ressources est les données de toutes les applications installées sur le système.

Les applications installées à partir de l'*App Store* sont chiffrées à l'aide du *DRM Fairplay*, leur analyse requiert leur déchiffrement sur le terminal, ce qui implique le *jailbreak* du terminal. Les applications ad-hoc n'étant pas protégées par le DRM, leur analyse reste possible sans recourir au *jailbreak.*

com.apple.mobile.installation_proxy : Le service installation proxy est utilisé lors des opérations de maintenance des applications [6]. Ce service supporte les commandes suivantes :

- Énumération des applications installées sur le terminal.
- Énumération des applications archivées par iTunes ;
- Archivage d'une application (création d'une archive ZIP dans le répertoire : « *ApplicationArchives* » et désinstallation de l'application) ;
- Suppression d'une application archivée
- ...

com.apple.mobilebackup & com.apple.mobilebackup2 : iTunes réalise les opérations de sauvegarde et récupération à l'aide des services *com.apple.mobilebackup* & *com.apple.mobilebackup2*

Lorsque l'iPhone est synchronisé sur un poste de travail, iTunes crée automatiquement un dossier dont le nom correspond à l'UDID de l'‘équipement (ID unique de l'équipement sous la forme de 40 caractères hexadécimaux). Il copie le contenu de l'appareil dans le dossier nouvellement créé. Cette synchronisation n'est réalisable qu'après déverrouillage de l'équipement et appairage avec le périphérique de sauvegarde. Comme le montre la figure 15, l'emplacement des sauvegardes depend du type du système.

Système	Emplacement des fichiers
Windows 7 & 8	`C:\Users\(username)\AppData \Roaming \Apple Computer \MobileSyntextbackslash Backup\`
OSX	`/Users/(username)/Library/Application Support/MobileSync/Backup/`

Figure 15. Emplacement des sauvegardes iOS

L'analyse des sauvegardes peut permettre à l'analyste de récupérer des données tel que le journal d'appel, les données des applications, et

6. Utilisé conjointement avec com.apple.mobile.house_arrest, ce service permet d'énumerer et de télécharger l'intégralité des applications installées sur le système à l'exception des applications systèmes d'Apple.

certaines informations stockées dans le coffre fort numérique du système. Cette opération peut être réalisée à l'aide des outils du framework iPhone-DataProtection [7] et de l'outil iPhoneBackupAnalyser2 [36]. L'extraction de données contenues dans les sauvegardes est conditionnée par le positionnement d'un mot de passe de sauvegarde par l'utilisateur.

com.apple.mobile.file_relay : Le service *file_relay* permet l'obtention de diverses bases de données et fichiers de configuration. Ce service supporte les commandes suivantes :
- SystemConfiguration ;
- UserDatabases ;
- CrashReporter ;
- Caches ;
- Tmp ;
- ...

Tous les fichiers renvoyés par l'iPhone sont stockés en clair dans une archive CPIO. Une fois extrait, ces fichiers se présentent sous la forme de bases de données sqlite. Les autres commandes, permettent d'obtenir les fichiers de configurations en cours d'utilisation sur le terminal.

com.apple.pcapd : L'analyse du trafic réseau peut se révéler être un atout lors de l'analyse d'un virus ou d'un programme malveillant. L'utilisation du service *com.apple.pcapd* peut répondre à ce besoin sans nécessiter la mise en place d'outillage complexe. En effet, ce service présent à partir de la version 5 d'iOS permet la création d'une interface TAP virtuelle sur l'équipement. Lorsque cette interface est active, tout le trafic réseau est recopié vers le poste d'analyse.

```
$ rvictl -s <[Device ID]>
Starting device <[Device ID]> [SUCCEEDED]
$ ifconfig -l
lo0 gif0 stf0 en0 en1 p2p0 fw0 ppp0 utun0 rvi0
$ sudo tcpdump  -i rvi0 -n
listening on rvi0, link-type RAW (Raw IP), capture size 65535 bytes
...
```

Listing 5. Activation de l'interface d'analyse du traffic réseau

Par conception, cette interface ne permet ni l'inspection des données transmises au travers de tunnel de communication chiffré (HTTPS, SSL, VPN...) ni l'analyse des échanges liés aux fonctionnalités téléphoniques (voix, sms...).

5.3 Conclusion

L'utilisation des services PTP et iTunes dans le but d'acquérir et d'analyser les données et les applications iPhone est une méthode qui n'entraine aucune modification du système.

Les fonctionnalités de sécurité du système ne sont donc pas désactivées ce qui interdit l'extraction des fichiers systèmes et l'analyse approfondie de l'environnement. Ces restrictions s'avèrent donc bloquantes dans le cadre de la recherche de virus ou de logiciel malveillant.

6 Extraction Logique de donnée (avec altération)

L'utilisation des services standard d'iTunes étant assez limitée en ce qui concerne l'analyse de virus ou logiciel malveillant, l'analyste peut rechercher d'autres techniques d'acquisition.

Cette section décrit l'utilisation d'outils et de techniques de jailbreak *dans le cadre d'une investigation numérique réalisée à la demande de l'utilisateur. Elle ne doit pas être vue comme une incitation à l'exploitation de vulnérabilité, ou à* jailbreaker *un terminal. Le* jailbreak *désactive les fonctionnalités de sécurité des terminaux augmentant ainsi la surface d'attaque [44] et le risque de compromission. Enfin, l'ANSSI [14] incite les éditeurs à corriger systématiquement toutes vulnérabilités identifiées, dans les plus brefs délais. Les utilisateurs sont quant à eux invités à appliquer ces correctifs dès leurs publications.*

6.1 Investigation numérique et *jailbreak*

La communauté *jailbreak* dispose d'outils permettant de contourner complètement les fonctionnalités de sécurité du système iOS [13]. Ces outils modifient de façon persistante, aussi bien le comportement du noyau que celui de divers composants du système des terminaux sur lesquels ils sont déployés. L'absence de documentation des éléments modifiés sur le terminal associé à la persistance de certaines modifications est incompatible avec les objectifs de l'investigation numérique.

L'analyse des différents outils de *jailbreak* publics (*evasi0n* [13], *p0sixpwn* [43] et [13]), met en évidence plusieurs techniques permettant de débrider partiellement un terminal Apple tout en minimisant l'impact sur le système.

6.2 Débridage du service AFC

L'activation temporaire d'une copie du service *com.apple.afc*, non soumise aux restrictions imposées par le système, permet d'acquérir l'intégralité des données stockées sur la NAND. L'impact de cette opération sur le système est variable en fonction de la version d'iOS cible. Les chapitres suivants présentent les actions requises pour débrider le service AFC sur les terminaux Apple.

Le cas d'iOS 6.x.x L'analyse du *Jailbreak p0sixpwn* met en évidence une vulnérabilité de type « race condition » permettant de contourner les restrictions imposées par la sandbox et ainsi modifier la configuration du système d'Apple. Il devient possible de débrider le service AFC en utilisant ce point d'entrée. Cette vulnérabilité initialement découverte par Nicholas Allegra *alias* Comex affecte un composant du service *com.apple.mobile.mobile_image_mounter* qui est utilisé par l'outil *Xcode* lors du déploiement des outils de développement sur un terminal de test.

Les outils de développement iOS se présentent sous la forme d'une image « dmg » signée par Apple. Lors du montage de ce disque sur le terminal, *installd* vérifie la validité de la signature et fusionne le contenu de l'image avec le contenu du disque système. *Installd* étant vulnérable à une *race condition* entre ces deux étapes il devient possible de monter un disque non signé sur le système.

L'utilisation de cette vulnérabilité dans le cadre d'une investigation requiert la création d'une image « dmg » contenant le fichier de configuration du service *AFC* débridé. Une fois le disque non signé monté, il est possible de démarrer le service et d'extraire l'intégralité du système de fichiers en utilisant les API de la bibliothèque *libimobiledevice*.

Le cas d'iOS 7.0 L'analyse du *Jailbreak evasi0n7* associée aux commentaires de George Hotz *alias* geohot [17] permettent d'identifier plusieurs vulnérabilités qui lorsqu'elles sont combinées autorisent l'activation d'un service *AFC* débridé.

Directory traversal dans installd : L'exploitation de cette vulnérabilité requiert l'utilisation d'une application signée par Apple. Une exploitation réussie permet d'obtenir un fichier disposant des droits de lecture, d'écriture et d'exécution dans un répertoire accessible par le service *afc*. Le contenu de ce fichier est ensuite modifié afin de permettre à l'analyste de lancer une version débridée du démon *afcd* en cliquant sur l'icône de l'application.

À ce stade, le service *afcd* est toujours soumis à la politique d'isolation du système. Il est donc nécessaire de la contourner.

Contournemant de la politque d'isolation d'iOS : Le service *afcd* initialise sa propre sandbox lors de son démarrage en appelant la bibliothèque (`/usr/lib/system/libsystem_sandbox.dylib`). Il est alors possible d'interposer une bibliothèque spécialement conçue pour interdire l'initialisation de la sandbox.

Contournement du contrôle de la signature du code : Pour que la bibliothèque soit chargée par le système il est nécessaire de contourner les restrictions liées à la signature obligatoire des exécutables. Ceci est réalisable en utilisant des *lazy bindings* [20,35] pour rediriger les fonctions d'initialisation et de vérification de l'état de la sandbox. Cette technique permet de réexporter des symboles sans utiliser de code. La réexportation des fonctions est réalisée dans l'entête Mach-O de l'exécutable et ne contient pas de code exécutable. Toutefois, pour que la bibliothèque soit valide et puisse être chargée par le système, l'éditeur de lien positionne le droit d'exécution sur la section TEXT du binaire.

```
$ dyldinfo -export gameover.dylib

for arch armv7:
export information (from trie):
[re-export] _SANDBOX_CHECK_NO_REPORT (_kCFBooleanTrue from
    CoreFoundation)
[re-export] _sandbox_check (_sync from libSystem)
[re-export] _sandbox_extension_consume (_sync from libSystem)
[re-export] _sandbox_extension_issue_file (_sync from libSystem)
[re-export] _sandbox_free_error (_sync from libSystem)
[re-export] _sandbox_init (_sync from libSystem)
[re-export] _sandbox_init_with_parameters (_sync from libSystem)

[..]
```

Listing 6. Reexportation de fonctions de gameover.dylib

À ce stade les fonctionnalités de vérification de la signature du code exécutable étant toujours actives, le chargement de cette bibliothèque non signée échoue. Il est donc nécessaire d'enlever les droits d'exécutions pour contourner les fonctionnalités de vérification de signature. L'utilisation de l'attribut `S_ATTR_LO_RELOC` [21] entraine la suppression de ce droit lors de l'analyse de l'entête du fichier. C'est à dire avant la projection des pages en mémoire et avant la vérification de la signature des sections exécutables.

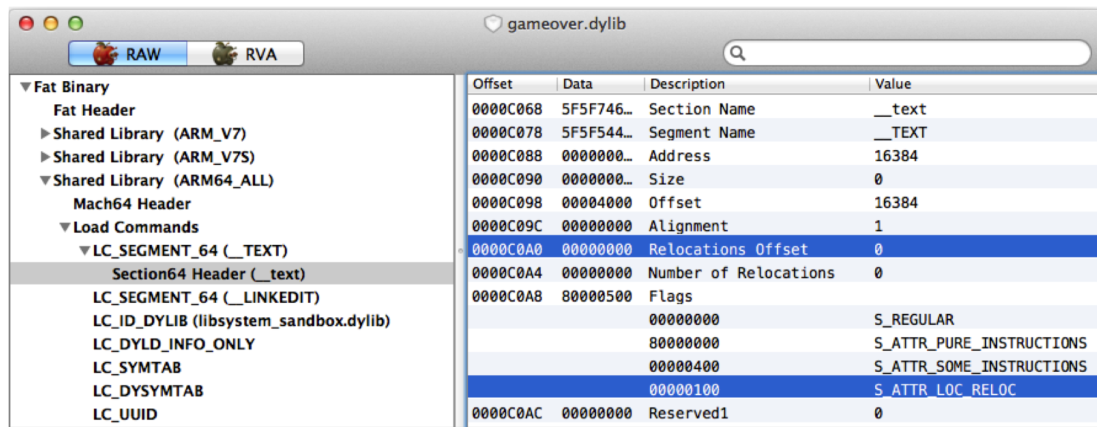

FIGURE 16. Analyse de la structure de gameover.dylib

Directory traversall & afcd : En plus des restrictions imposées par les fonctionnalités d'isolation d'iOS, le service *afcd* intègre des fonctions en charge d'interdire le *directory traversal*. Ainsi, il ne peut accéder qu'au contenu du répertoire spécifié lors de son démarrage : `/var/mobile/Media`. Cependant, l'implémentation de cette fonctionnalité reste perfectible. Lors de la création d'un lien symbolique, *afcd* compte le nombre de « `../` » ce qui a pour effet d'interdire l'accès aux fichiers situés à l'extérieur du répertoire racine à partir duquel le service est démarré. Cependant, le simple déplacement du lien dans un répertoire de niveau inférieur permet de contourner ce mécanisme de protection.

Chargement du code de contournement de la sandbox : Il reste maintenant à forcer le chargement de cette bibliothèque au démarrage du service. Cette opération peut être réalisée en spécifiant la variable d'environnement `DYLD_INSERT_LIBRARY` (équivalent de `LD_PRELOAD` sous Linux) dans les paramètres de démarrage de l'application signée, précédemment installée sur le terminal. Ces paramètres sont spécifiés dans le fichier `/private/var/mobile/Library/Cache/com.apple.mobile.installation.plist`. Ce fichier n'est pas accessible par le service *com.apple.afc*. Toutefois, il est possible d'en obtenir une copie en utilisant le service `com.apple.mobile.file_relay` avec la commande `Caches`. Une fois le fichier modifié il faut le renvoyer sur le système et forcer son rechargement.

Race condition & installd : Sur le système iOS, le binaire *installd* n'est pas soumis aux règles d'isolation imposées par la sandbox. Or, cet exécutable est affecté par une vulnérabilité de type *race condition* qui autorise l'extraction

de fichier ZIP sur le système. L'exploitation de cette vulnérabilité requiert néanmoins un accès au répertoire `/tmp`. Cet accès peut être obtenu par l'exploitation de la vulnérabilité décrite dans le paragraphe précédant. Il est donc possible de remplacer le fichier *com.apple.mobile.installation.plist* par sa version modifiée.

L'extraction du contenu de la charge active dans le répertoire : `../../../var/mobile/Library/Caches/` permet d'actualiser le *com.apple.mobile.installation.plist* et de forcer son rechargement au prochain démarrage du système (création du fichier). Cette action permet également de supprimer la limite de temps allouée au démarrage d'une application.

Rechargement du caches : Le rechargement des différents caches est obtenu par le redémarrage du système à l'aide du service *com.apple.diagnostique_relay* et de la commande « *restart* ».

Débridage du service AFC : Lorsque l'analyste lance l'application, le système lance un second service *afcd* exécuté cette fois à l'extérieur de la *sandbox*. À ce stade la partition système est toujours en lecture seule. Néanmoins, l'analyste dispose d'un accès complet au système de fichiers.

7 Limites des méthodes d'acquisition

L'acquisition physique des données n'entraine aucune modification sur le terminal et permet la récupération du contenu de certains fichiers effacés. Cette méthode est donc à privilégier lors de la recherche de preuves dans le cadre d'une analyse réalisée pour confirmer ou infirmer une compromission. Toutefois, cette méthode est basée sur une vulnérabilité affectant la chaîne de démarrage des terminaux équipés d'un processeur Apple A4 ou antérieur. Il est donc Impossible d'utiliser cette technique sur les terminaux récents. La figure 17 présente la compatibilité des méthodes d'acquisition avec les différents terminaux Apple.

Modèle	Acquisition physique (sans *jailbreak*)	Acquisition logique
iPhone 4 & inférieur	Oui	Oui (équipement déverrouillé)
iPhone 4s & supérieur	Non	
iPad 1	Oui	
Pad 2 & supérieur	Non	

Figure 17. Compatibilité du matériel avec les méthodes d'acquisition

La figure 18 présente la compatibilité des méthodes d'acquisition avec les différentes versions d'iOS. La méthode d'acquisition logique « sans altération », bien que compatible avec toutes les versions de terminaux en circulation, interdit l'export complet du système de fichiers. Cette méthode est donc inadaptée dans le cadre de la recherche de preuves permettant d'infirmer ou confirmer une compromission.

Modèle	Acquisition logique sans altération	Acquisition logique avec altération
iOS $< 7.0.6$	Oui	Oui
iOS $> 7.0.6$		Non

FIGURE 18. Compatibilité du système avec les méthodes d'acquisition

En s'appuyant sur des vulnérabilités, la méthode d'acquisition logique « avec altération » permet d'obtenir une image du système d'un terminal compatible. Toutefois, cette technique entraine des modifications sur le système. Il est donc essentiel de contrôler et tracer ces modifications car celles-ci peuvent influer sur les résultats d'une investigation.

Apple corrige régulièrement les vulnérabilités affectant ses terminaux. L'utilisation de vulnérabilités afin d'autoriser l'acquisition du système de fichiers n'est donc pas une technique pérenne.

Enfin, toutes ces méthodes se concentrent sur l'extraction du système de fichiers. L'acquisition et l'analyse de la mémoire RAM ainsi que l'analyse du code exécuté par le « Baseband » du terminal ne sont aujourd'hui pas réalisables sans le débridage complet du terminal.

8 Mise en pratique

Lors d'une analyse *forensic* il est important de concentrer les ressources d'analyse sur les éléments pertinents dans le contexte. La génération de la liste des fichiers légitimes qui doivent être présents sur le système (liste blanche) et la liste des fichiers correspondant aux menaces connues (liste noire) est donc nécessaire. Seule la création de la liste blanche est abordée dans cet article. La récupération des différents échantillons de *malware* ou d'outils de *jailbreak* nécessaires à la création d'une liste noire est laissée en exercice au lecteur.

8.1 Acquisition d'une image de référence (Liste blanche)

Il existe deux méthodes pour obtenir cette image de référence : depuis un fichier *IPSW* ou depuis un terminal de référence après débridage du service *AFC*.

Obtention d'un système de fichiers de référence depuis un fichier IPSW Toutes les versions existantes des images de *firmwares* sont téléchargeables sur le site d'Apple à condition de connaître l'URL d'accès direct. Une liste des différentes URL est maintenue à jour sur l'iPhone Wiki [28] par la communauté *jailbreak*.

Architecture d'une image de firmware Apple : Une image de *firmware* Apple iOS se présente sous la forme d'un fichier iPhone Software (IPSW). Ce fichier est une archive au format ZIP.

Non du fichier	Fonction	Type de conteneur
batterylow0*.img3 batterylow1*.img3 batteryfull*.img3 batterycharging0*.img3 batterycharging1*.img3 glyphcharging*.img3 glyphplugin*.img3	Icones indiquant le niveau de batterie ou l'état de charge	IMG3
DeviceTree.<board>.img3	Device tree	IMG3
iBoot.<device>ap.RELEASE.img3	iBoot	IMG3
LLB.<device>ap.RELEASE.img3	Low level boot loader (LLB).	IMG3
kernelcache.release.<device>	Noyau & Extension noyau	IMG3
applelogo*.img3	Logo Apple	IMG3
recoverymode*.img	Image de récupération	IMG3
xxx-«lowest numbered»-yyy.dmg	Système de fichiers racine	DMG FileVault
xxx-«middle numbered»-yyy.dmg	Ramdisk de mise à jours	IMG3
xxx-«highest numbered»-yyy.dmg	Ramdisk de récupération	IMG3

Figure 19. Description des compsants d'une image IPSW

Après extraction, l'analyse du fichier *BuildManifest.plist* permet d'identifier les fichiers contenant l'image du système de fichiers racine : *058-2399-001.dmg* et l'image du noyau : *kernelcache.release.n90*. Apple a intégré des mécanismes de sécurité permettant de s'assurer de l'intégrité et de l'authenticité de l'image du *firmware* avant réinstallation. Ces mécanismes sont notamment basés sur le chiffrement et la signature des différents composants embarqués dans l'image du *firmware* (fichier IPSW). À l'exception de l'image du système qui est stockée au format DMG, les fichiers

stockés dans l'archive IPSW d'un *firmware* Apple sont au format IMG3. Chacun de ces fichiers est chiffré avec sa propre clé de chiffrement, elle-même chiffrée par la clé GID enfouie dans le processeur cryptographique du terminal.

Apple Image file (IMG3) : Les fichiers IMG3 servent de conteneurs chiffrés. Le format est relativement simple. Ils commencent par une entête dont la structure est présentée sur le listing 7. Cette entête contient : un identifiant (*magic*) : `IMG3`, la taille total du fichier et la taille des données avant compression.

```
typedef struct {
    unsigned int magic;
    unsigned int full_size;
    unsigned int data_size;
    unsigned int shsh_offset;
    unsigned int image_type;
} img3_header;
```

Listing 7. Structure de l'entête d'un fichier IMG3

À la suite de cette entête sont accolées différentes sections dont la structure est présentée sur le listing 8.

```
typedef struct {
    unsigned int tag_name;
    unsigned int tag_full_size;
    unsigned int tag_data_size;
} img3_tag;
```

Listing 8. Structure d'une section de fichier IMG3

Comme pour la plupart des formats propriétaires d'Apple, les données sont organisées suivant l'orientation *little-endian*. Chaque section débute par un mot clé de quatre octets identifiant le contenu de la section. La figure 20 présente les sections identifiées lors de l'analyse du fichier *kernelcache.release.<device>* contenant le noyau XNU.

TYPE	Type de l'image
DATA	Contenu réel du fichier
SEPO	Security Epoch
KBAG	Clé AES et l'IV qui ont servi à chiffrer les données
SHSH	Empreinte SHA-1 Chiffrée avec la clé privée d'Apple
CERT	Certificat d'Apple

Figure 20. Sections d'un fichier kernelcache.release.<device>

Les éléments permettant de déchiffrer chacun des fichiers `IMG3` se trouvent dans la section `KBAG`. Cette section contient la clé AES ainsi que l'IV qui ont servi à chiffrer le contenu de la section `DATA`. Toutefois, ces éléments cryptographiques sont eux même chiffrés (AES256) avec la clé GID enfouie dans le processeur cryptographique.

```
typedef struct Kbag_t {
    u32  magic;       //(KBAG)
    u32  totalLength;
    u32  dataLength;
    int  cryptState; // 1 = encrypted
    int  aesType;    // 0x80 = aes128/0xc0 = aes192/0x100 = aes256
    char encIV[16];  // IV
    char encKey[16/24/32]; // Key length = 128, 192 or 256 bits
} Kbag;
```

Listing 9. Structure du KBAG d'un fichier IMG3

L'utilisation de la clé de chiffrement GID propre à chaque famille d'équipement est donc obligatoire pour déchiffrer le contenu des éléments stockés dans l'image IPSW. Or cette clé est uniquement accessible par le processeur cryptographique lors du démarrage du système. Il est donc nécessaire d'avoir recours à l'exploitation d'une vulnérabilité pour déchiffrer les différentes images [7] [8].

Une fois les clés de déchiffrement obtenues, il est possible de déchiffrer le noyau à l'aide de deux outils disponibles publiquement : *xpwn* [37] et *lzssdec* [16]. Le noyau obtenu après déchiffrement et décompression peut servir de référence pour l'analyse d'un système compromis.

Disc iMaGe files (DMG) : L'image dans laquelle est stockée le système de fichiers racine est une image au format DMG. Ce format est utilisé pour stocker des systèmes de fichiers complets (habituellement HFS+) dans un seul fichier. Les images DMG fonctionnent comme des disques physiques réels. Le format supporte le stockage de données brutes (RAW), compressées ou chiffrées (Utilisation des algorithmes de chiffrement FileVault d'Apple).

Dans le cas de l'image du système de fichiers racine d'un système iOS, l'image est découpée en blocs (*chunk*) de 512 octets. Chaque bloc est

7. À la date de rédaction de cet article, il n'est pas possible d'obtenir les clés de déchiffrement des terminaux récents (IPhones 5, 5c, 5s) à l'aide d'outils publics.

8. La communauté *jailbreak* à mis en place sur l'iPhone Wiki [27] une page sur laquelle sont centralisées toutes les clés de déchiffrements de *firmware* d'équipement affectés par une vulnérabilité dans leur chaîne de démarrage. Il est conseillé de surveiller cette page pour obtenir les clés de chiffrement utilisées pour les nouveaux firmwares.

chiffré (AES128) avec une clé et un vecteur d'initialisation générés à partir de la clé *VFDecrypt* [3].

$$aes_key = VFDecryptKey[0:15]$$
$$hmac_key = VFDecryptKey[15:]$$
$$IV = trunc128(HMACSHA1(hmac_key||chunkno))$$

La clé VFDecrypt chiffrée (DES-EDE3-CBC) à l'aide de la clé ASR qui est stockée dans l'entête de l'image « dmg ». Il existe une clé ASR pour chaque type de plateforme. L'algorithme de génération de cette clé peut varier suivant les versions d'iOS. Lors de la restauration du système, l'image est décompressée à l'aide de la clé ASR générée par le binaire `/usr/sbin/asr` qui est contenu dans l'image de restauration du système. L'image de restauration du système étant au format IMG3 chiffré, il est nécessaire d'avoir recours à l'exploitation d'une vulnérabilité dans la chaîne de démarrage terminal pour la déchiffrer. Cette approche est donc limitée aux plateformes vulnérables [9].

Une fois la clé VFDecrypt récupérée il est possible d'extraire l'image DMG. Sous OSX l'extraction est réalisée à l'aide de la commande *hdiutil*, en utilisant l'option de montage `-ImageKey DiskImage -class = CRawDiskImage`.

```
hdiutil attach ./058-2399-001.dmg -imagekey diskimage-class=
    CRawDiskImage
```

Listing 10. Déchiffrement de l'image du système de fichiers (OSX)

Sur les autres systèmes, l'extraction requiert l'utilisation des outils *vfdecrypt* [2], *dmg2img* [8] et *mount*.

```
$ vfdecrypt -i./058-2399-001.dmg -k<key> -o<output.dmg

$ dmg2img <output>.dmg

$ sudo mount -t hfsplus -o loop <output>.dmg /mnt/ios_rootfs/
```

Listing 11. Déchiffrement de l'image du système de fichiers (Linux)

Une fois le système de fichiers de référence monté il ne reste plus qu'à accéder au système de fichiers du terminal et de procéder à une analyse différentielle entre le contenu des images de références et celle du sytème en cours d'analyse.

9. Les clés VFDecrypt de chacun des *firmware* sont également présentes sur la page de centralisation des clés de chiffrement des *firmware* de l'iPhone Wiki [27].

Conclusion : L'utilisation du processeur cryptographique d'un équipement de la même famille que le terminal à analyser est obligatoire afin d'obtenir les clés de déchiffrement des différents composants du firmware. L'utilisation du processeur cryptographique n'est possible que lors du démarrage du système. Il est donc nécessaire d'avoir recours à l'exploitation d'une vulnérabilité dans la chaîne de démarrage. À la date de rédaction de cet article seuls les équipements équipés de puce A4 sont exploitables à l'aide d'outils publique tel que *limera1n* et *OpeniBoot*.

Extraction d'une image de référence à partir d'un équipement de confiance Le déchiffrement des fichiers contenus dans les archives IPSW n'étant pas réalisable pour les terminaux récents il est nécessaire de trouver une méthode alternative. La méthode présentée dans la section 6, permettant d'obtenir une image du système de fichiers, il suffit de l'exécuter sur un terminal de référence pour obtenir l'image qui servira de base à l'analyse. Toutefois, cette méthode est elle-même limitée au terminaux vulnérables.

Comparaison de système de fichiers Lors d'une analyse *forensics*, il est intéressant d'identifier la présence de fichiers connus. Cette opération peut être réalisée par la création de liste blanche et/ou liste noir. Par exemple en calculant l'empreinte de chacun des fichiers présents sur un terminal de référence et en répétant cette opération sur le terminal en cours d'analyse. Cette tâche peut être automatisée à l'aide d'un outil tel que *hashdeep* [29].

Context Triggered Piecewise Hashing (CTPH) : Un algorithme tel que MD5 ou SHA1 ne permet pas d'identifier les fichiers partageants du contenu commun avec un fichier de référence. Ce qui peut être utile pour identifier des menaces connues ayant été plus ou moins modifiées. En 2006 Jesse Kornblum a publié un article intitulé : « *Identifying almost identical files using context triggered piecewise hashing* » [30] dans lequel il présente l'algorithme CTPH. Cet algorithme initialement développé par Andrew Tridgell dans le but de détecter le *spam* a été transposé au monde du *forensics* par Jesse Kornblum sous la forme de l'outil *ssdeep* [31]. Contrairement aux outils de hachages classiques, *ssdeep* permet d'identifier des fichiers homologues. Un exemple de résultat est présenté sur le listing 12.

```
$ ls -l foo
-rw-rw---- 1 mre staff 63284 5 oct 22:40 foo
$ cp foo bar
```

```
$ echo 1 >> bar
$ ssdeep -b foo > hashes
$ cat hashes
ssdeep,1.1--blocksize:hash:hash,filename
1536:fSUBu6FV47jOEIQlKTDReTf1hJm+zFRjGe:qqXQyUGe,"foo"

$ ssdeep -bm hashes bar
bar matches hashes:foo (99)
```

Listing 12. ssdeep : Comparaison de deux versions d'un même fichier

Le nombre à la fin de la ligne est un score qui indique le pourcentage de similarité entre les deux fichiers comparés. Plus ce nombre est élevé plus le taux de correspondance entre deux fichiers est important. Il est donc très simple d'identifier des fichiers homologues.

Identification des modifications apportées par le Jailbreak : En 2013 Bernardo Rodrigues a intégré l'algorithme CTPH sous forme d'un script python. *Binwally* [38] utilise la bibliothèque python *ssdeep* pour analyser les différences entre deux arborescences de répertoires afin d'identifier les différences liées à l'ajout, la suppression ou la modification de certains fichiers. Les résultats retournés par l'outil incluent un score global permettant d'identifier l'importance des éventuelles modifications. L'utilisation de cet outil, sur l'arborescence d'un système jailbreaké et l'arborescence d'un système de référence de la même famille, permet d'identifier les fichiers suspects présentés sur le listing 13.

```
25 differs etc/fstab
25 differs private/etc/fstab
>>> unique  target/var/mobile/Media/.evasi0n7_installed
>>> unique  target/var/mobile/Media/jailbreak.log
>>> unique  target/private/var/tmp/evasi0n-started}
>>> unique  target/sbin/dmesg
>>> unique  target/sbin/dynamic_pager
>>> unique  target/sbin/halt
>>> unique  target/sbin/nologin
>>> unique  target/sbin/reboot
[..]
>>> unique  target/System/Library/LaunchDaemons/com.evad3rs.evasi0n7
    .untether.plist
>>> unique  target/System/Library/LaunchDaemons/com.saurik.Cydia.
    Startup.plist
>>> unique  target/System/Library/Caches/com.apple.xpcd/xpcd_cache.
    dylib
>>> unique  target/System/Library/Caches/com.apple.dyld/enable-
    dylibs-to-override-cache
[..]
>>> unique  target/tmp/evasi0n-started
>>> unique  target/usr/bin/apt-key
>>> unique  target/usr/bin/arch
>>> unique  target/usr/bin/bashbug
```

```
>>> unique  target/usr/bin/captoinfo
>>> unique  target/usr/bin/cfversion
>>> unique  target/usr/bin/clear
>>> unique  target/usr/bin/cmp
[..]
>>> unique  target/usr/lib/_ncurses/
[..]
>>> unique  target/usr/lib/libmenu.5.dylib
>>> unique  target/usr/lib/libmenu.dylib
>>> unique  target/usr/lib/libmenuw.5.dylib
>>> unique  target/usr/lib/libmenuw.dylib
>>> unique  target/usr/lib/libmis.dylib
[..]
>>> unique  target/usr/lib/apr.exp
>>> unique  target/usr/lib/apt/methods/bzip2
[..]
>>> unique  target/usr/sbin/vipw
>>> unique  target/pri/vate/var/mobile/Media/.evasi0n7_installed
>>> unique  target/private/var/mobile/Media/jailbreak.log
>>> unique  target/evasi0n7
>>> unique  target/evasi0n7-installed}
>>> unique  target/bin/bash
>>> unique  target/bin/bunzip2
>>> unique  target/bin/bzcat
[..]
```

Listing 13. Fichiers suspects sur un terminal Jailbreaké

Les résultats présentés sur le listing 13 n'incluent pas l'analyse des *du contenu de* `/dev`. Il faut donc procéder à une analyse manuelle pour identifier les différences entre les deux systèmes. La comparaison des répertoires met en évidence sur le listing 14, deux périphériques supplémentaires.

```
/dev/hax-ptsd
/dev/hax-test
```

Listing 14. Présence d'éléments suspects dans la liste de périphériques

Les résultats présentés sur le listing 15 montrent qu'aucune modification n'a été portée au noyau présent sur le système de fichiers du terminal en cours d'analyse. Ce qui est normal car *evasi0n* modifie le noyau en mémoire lors du démarrage du terminal.

```
100 matches System/Library/Caches/com.apple.kernelcaches/kernelcache
```

Listing 15. Résultats de l'analyse de l'image du noyau

Ces résultats permettent de valider le bon fonctionnement des outils d'acquisition. L'utilisation de l'algorithme CTPH permet de réduire la surface d'analyse. Toutefois, une analyse manuelle reste nécessaire pour identifier l'intégralité des éléments suspects. Enfin, une phase d'analyse

approfondie est indispensable pour comprendre le fonctionnement des éléments identifiés. Le but de cet article n'étant pas d'expliquer le fonctionnement du *evasi0n*, ce point ne sera pas abordé.

9 Conclusion

Les terminaux Apple sont des ordinateurs à part entière. L'usage de ces équipements à des fins personnelles et professionnelles en fait une cible de choix pour les attaquants. Le système iOS intègre un grand nombre de fonctionnalités de sécurité. Cependant, il existe aujourd'hui une version de Jailbreak pour chaque version majeure d'iOS, montrant ainsi que ce système reste perméable. Or, Apple n'a prévu aucune fonctionnalité autorisant l'analyse de l'intégrité de son système d'exploitation mobile. Les méthodes d'acquisition de données présentées dans ce document comblent en partie ce manque. Cependant certaines de ces techniques sont basées sur des vulnérabilités et ne sont pas pérennes.

Références

1. Universal serial bus device class specification for device firmware upgrade. `http://www.usb.org/developers/devclass_docs/usbdfu10.pdf`, 1999.
2. Anonyme and Drake Allegrini. Vfdecrypt. `https://github.com/fswrangler/VileFault/tree/master/vfdecrypt`, 2010.
3. Jacob Appelbaum and Ralf-Philipp Weinmann. Unlocking filevault : An analysis of apple's encrypted disk storage system. `http://events.ccc.de/congress/2006/Fahrplan/attachments/1244-23C3VileFault.pdf`, 2006.
4. Nikias Bassen. libimobiledevice & usbmuxd. `http://www.libimobiledevice.org/`, 2010.
5. Andrey Belenko. Overcoming ios data protection to re-enable iphone forensics. `https://media.blackhat.com/bh-us-11/Belenko/BH_US_11_Belenko_iOS_Forensics_Slides.pdf`, `http://2011.brucon.org/images/2/28/Brucon2011-Belenko_-_iOS_Data_Protection.pdf`, 2011.
6. M.L.H. Brouwer and M.D. Adler. System and method for content protection based on a combination of a user pin and a device specific identifier, October 13 2011. US Patent App. 12/797,587.
7. Jean-Baptiste Bédrune and Jean Sigwald. iphonedataprotection framework (code source). `https://code.google.com/p/iphone-dataprotection/`, 2010.
8. Jean-Pierre Demailly and vu1tur. Dmg2img. `http://vu1tur.eu.org/tools/`, 2007.
9. Mark Dowd and Tarjei Mandt. ios 6 kernel security : A hacker's guide. `http://conference.hitb.org/hitbsecconf2012kul/materials/D1T2-MarkDowd\&TarjeiMandt-iOS6Security.pdf`, 2012.
10. Stefan Esser. ios kernel exploitation iokit edition. `http://reverse.put.as/wp-content/uploads/2011/06/SyScanTaipei2011_StefanEsser_iOS_Kernel_Exploitation_IOKit_Edition.pdf`, 2011.

11. Stefan Esser. ios 5 – an exploitation night mare. http://antid0te.com/CSW2012_StefanEsser_iOS5_An_Exploitation_Nightmare_FINAL.pdf, 2012.
12. Stefan Esser. Mountain lion ios vulnerabilities garage sale. http://www.syscan.org/index.php/download/get/1181eec74aeb7d5d7c7b728dbd823e6f/SyScan2013_DAY2_SPEAKER12_Stefan_Esser_Mountain_Lion_iOS_Vulnerabilities_Garage_Sale.zip, 2013.
13. evad3rs. Evasi0n jailbreak. http://evasi0n.com/, 2012.
14. Secrétariat général de la défense et de la sécurité nationale. Recommandations de sécurité relatives aux ordiphones. http://www.ssi.gouv.fr/IMG/pdf/NP_Ordiphones_NoteTech.pdf, 2013.
15. Cédric Halbronn and Jean Sigwald. iphone data protection : iphone security model & vulnerabilities. http://eseclab.sogeti.com/dotclear/public/publications/10-hitbkl-iphone.pdf, 2010.
16. Willem Hengeveld. lzssdec. http://nah6.com/~itsme/cvs-xdadevtools/iphone/tools/lzssdec.cpp, 2009.
17. George Hotz. evasi0n7 writeup. http://geohot.com/e7writeup.html, 2013.
18. George Hotz and Chronic-Dev Team. Exploit limera1n (code souce). https://github.com/Chronic-Dev/syringe/blob/master/syringe/exploits/limera1n/limera1n.c, http://limera1n.com/, 2010.
19. Communauté idroid. Openiboot. http://www.idroidproject.org/wiki/OpeniBoot, 2011.
20. Apple Inc. Executing mach-o files. https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/MachOTopics/Mach-O_Programming.pdf, 2009.
21. Apple Inc. Os x abi mach-o file format reference. https://developer.apple.com/library/mac/documentation/developertools/conceptual/machoruntime/Mach-O_File_Format.pdf, 2009.
22. Apple Inc. Projets open sources apple. http://www.opensource.apple.com, 2009.
23. Apple Inc. ios security. http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf, 2012.
24. iPhone Dev Team. redsnow. http://blog.iphone-dev.org/, 2008.
25. Communauté Jailbreak. Recovery mode (protocols) and dfu 0x1227. http://theiphonewiki.com/wiki/Recovery_Mode_(Protocols), http://theiphonewiki.com/wiki/DFU_0x1227.
26. Communauté Jailbreak. Usbmux. http://theiphonewiki.com/wiki/Usbmux.
27. Communauté Jailbreak. Liste des clés de déchiffrement des firmwares ios. http://theiphonewiki.com/wiki/Firmware_Keys, 2010.
28. Communauté Jailbreak. Liste des firmwares ios. http://theiphonewiki.com/wiki/Firmware, 2010.
29. Jesse Kornblum. hashdeep. http://md5deep.sourceforge.net/, 2003.
30. Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3 :91–97, 2006.
31. Jesse Kornblum. ssdeep. http://ssdeep.sourceforge.net/, 2006.
32. Tarjei Mandt. Attacking the ios kernel : A look at "evasi0n". http://www.nislab.no/content/download/38610/481190/file/NISlecture201303.pdf, 2013.

33. Charlie Miller. Breaking ios code signing. http://reverse.put.as/wp-content/uploads/2011/06/syscan11_breaking_ios_code_signing.pdf, 2011.
34. Charlie Miller, Dion Blazakis, Dino Dai Zovi, Stefan Esser, Vincenzo Iozzo, and Ralf-Phillip Weinmann. *iOS Hacker's Handbook*. John Wiley & Sons, Inc, 2012.
35. Camille Mougey. Evasi0n jailbreak : Precisions on stage 3. http://blog.quarkslab.com/evasi0n-jailbreak-precisions-on-stage-3.html, 2013.
36. Mario Piccinelli. iphonebackupanalyser. http://www.ipbackupanalyzer.com/, 2012.
37. planetbeing. xpwn. https://github.com/planetbeing/xpwn, 2008.
38. Bernardo Rodrigues. binwally. http://w00tsec.blogspot.fr/2013/12/binwally-directory-tree-diff-tool-using.html, 2013.
39. Jean Sigwald. iphonedataprotection building custom ramdisk & kernel. https://code.google.com/p/iphone-dataprotection/wiki/README, 2010.
40. Jean Sigwald. Apple key store kdf/tangling. https://code.google.com/p/iphone-dataprotection/source/browse/ramdisk_tools/AppleKeyStore_kdf.c, 2011.
41. Jason Tanner, Jim Morrison, Dick James, Ray Fontaine, and Phil Gamache. Inside the iphone 5s. http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-iphone-5s/, 2013.
42. Winocm. opensn0w (code source). https://github.com/winocm/opensn0w, 2012.
43. Winocm. p0sixspwn (code source). https://github.com/p0sixspwn, 2013.
44. Winocm. Evading ios security. http://winocm.com/projects/research/2014/01/11/evading-ios-security/, 2014.