PoC(k)ET, les détails d'un rootkit pour Windows Mobile 6

Cédric Halbronn
cedric.halbronn(@)sogeti.com

Sogeti / ESEC R&D

Résumé Les terminaux mobiles sont omniprésents dans notre monde actuel sous diverses formes : GPS, téléphones portables, PDAs, etc. Le smartphone est le résultat de la convergence de toutes ces plateformes. Il existe de nombreux systèmes d'exploitation sur le marché mobile. Windows Mobile [1], développé par Microsoft, est assez répandu [2].

Les mécanismes permettant de compromettre un PC sous Windows et d'y installer des portes dérobées (backdoors) sont connus depuis de nombreuses années. La version embarquée du système de Microsoft semble très similaire en surface puisque de nombreuses APIs qui existent dans les versions PC de Windows sont également présentes dans Windows Mobile. Cependant, les couches sous-jacentes sont très différentes : architecture matérielle ARM [3], gestion de la mémoire, appels systèmes, etc.

Nous détaillerons les façons d'injecter un rootkit, les mécanismes de furtivité (hook d'APIs, insertion de certificat, etc.), les possibilités de contrôle à distance (3G, Bluetooth, Wi-Fi, ActiveSync, interception de SMS, etc.), les façons de rendre le rootkit permanent, et les services (téléphone, SMS, GPS, SD-card, etc.) qu'un attaquant pourrait utiliser sur des terminaux sous Windows Mobile 6. Les mécanismes de sécurité implémentés par Microsoft seront analysés et comparés aux risques. Nous évaluerons enfin le niveau de protection fourni par les solutions antivirus existantes.

1 Introduction

1.1 Contexte

Le "smartphone" Le nombre de smartphones dans le monde croît de jour en jour. Ils correspondent à une évolution du téléphone portable auquel de nombreuses fonctionnalités sont ajoutées : PDA, Wi-Fi, GPS, etc. Par conséquent, un véritable système d'exploitation permettant de gérer cette multitude de services est nécessaire. Les systèmes d'exploitation principaux des smartphones actuels sont

Symbian OS (Nokia principalement), RIM OS (BlackBerry), Windows Mobile (nombreux constructeurs parmi HTC, HP, etc.), iPhone OS, Android (Google) ou encore Maemo (Nokia). Alors qu'ils deviennent de plus en plus complexes, peu d'études ont permis d'évaluer leur niveau de sécurité actuel.

Cette multitude de services augmente l'intérêt que peut avoir un attaquant de compromettre ces plateformes. En effet, ils contiennent un gestionnaire de contacts, navigateur web, lecteur e-mail, SMS, GPS, etc. De plus, ils permettent à l'utilisateur d'être connecté à des réseaux variés : Wi-Fi, GPRS, UMTS, Bluetooth, connexion avec un PC. Par conséquent, cela créé autant de vecteurs d'attaque potentiellement exploitables par une personne malveillante.

Windows Mobile 6 est la version du système d'exploitation de Microsoft actuellement utilisée sur les smartphones. Un point important à souligner est le fait qu'il est relativement aisé de porter une application depuis Microsoft Windows (XP, 2000 ou Vista), car un sous ensemble des APIs Win32 existe sous Windows Mobile. En règle générale, les APIs sont les mêmes, mais certains paramètres ont des valeurs positionnées automatiquement par Windows CE et ne sont donc pas utilisables par le développeur, ce qui finalement simplifie le développement. De plus, la plupart des ordinateurs de bureau attaqués utilisent le système d'exploitation Windows. Notons enfin que les constructeurs de smartphones ont pour la plupart choisi l'architecture ARM qui semble répondre aux besoins et contraintes des systèmes embarqués.

Notre étude se focalisera donc sur les smartphones utilisant l'architecture ARM et le système Windows Mobile 6. L'objectif de notre étude est de mettre en place un rootkit pour smartphone sous Windows Mobile 6 capable d'enregistrer des informations, puis de les exfiltrer. Nous supposerons que l'attaquant a un accès physique au smartphone pour y injecter le rootkit. Nous tenterons d'évaluer les mécanismes de furtivité disponibles, les possibilités de contrôler le rootkit à distance et de rendre le rootkit persistant à un redémarrage du terminal. Les nombreux services offerts par les smartphones actuels seront analysés afin d'évaluer dans quelle mesure un attaquant peut les utiliser à des fins malveillantes.

1.2 Objectifs

Qu'est-ce qu'un rootkit ? À son origine, le terme "rootkit" était utilisé pour désigner un ensemble de logiciels permettant de garder un accès "root" ou "administrateur" sur un ordinateur à l'insu de son utilisateur légitime. Cela signifie qu'un rootkit ne fournit pas de droits privilégiés et qu'il est nécessaire de les posséder avant l'installation du rootkit.

Le sens du terme "rootkit" a évolué depuis son apparition. En effet, les premiers rootkits étaient facilement détectables. Les attaquants ont alors implémenté des techniques permettant de masquer leur présence sur l'ordinateur cible. Ainsi, nous pouvons définir le terme "rootkit" comme un outil qui permet de masquer à l'utilisateur légitime du PC le fait que son système soit compromis dans le but de maintenir un contrôle total sur ce système.

Nous pouvons définir un "rootkit" sous Windows Mobile comme un ensemble de logiciels qui permet à un attaquant d'exécuter des commandes à distance et d'expatrier des informations sans que l'utilisateur du smartphone n'en soit notifié.

Composants "classiques" Un rootkit est typiquement composé des modules suivants [4] :

- Injecteur : ce module est relatif aux vecteurs d'attaque et permet d'installer le rootkit ;
- Protection : ce module correspond au "cœur" du rootkit. Il s'occupe de masquer sa présence à l'utilisateur, de le rendre persistant à un redémarrage de l'ordinateur et de le rendre résistant à toute action tentant de le supprimer ;
- Backdoor : il s'agit du canal de communication utilisé par l'attaquant pour contrôler à distance le rootkit et des vecteurs utilisés sur la cible pour exécuter les services ;
- Services : ce sont les services qui intéressent l'attaquant.

Contraintes à prendre en compte

L'embarqué / l'environnement mobile. Windows Mobile 6, même s'il apparaît similaire en surface aux versions PC de Windows, a de nombreuses différences en interne. La gestion de la mémoire, le fonctionnement des appels système sont notamment différents. Ceci doit

être pris en compte par un logiciel malveillant sans quoi la durée de vie de la batterie de l'appareil mobile serait grandement réduite. Ceci dévoilerait la présence du logiciel malveillant à l'utilisateur légitime du smartphone.

Les smartphones ne sont en général pas connectés 24 h/24 au réseau de l'opérateur, comme peuvent l'être les PC ou serveurs connectés à Internet. Ceci est notamment dû à la couverture du réseau [1] qui n'est pas uniforme sur l'ensemble du territoire. Ceci permet également au smartphone d'économiser sa batterie. Ainsi, un logiciel malveillant installé sur un smartphone doit prendre en compte cet environnement.

Les services offerts. Le smartphone est l'exemple par excellence de la convergence. Il regroupe les éléments de plusieurs systèmes embarqués : téléphone, PDA, GPS, appareil photo, microphone, etc. ainsi que des services nés du monde des PC : client Wi-Fi, navigateur Web, etc. Les services disponibles sont au final plus variés que sur un PC et un attaquant peut en bénéficier directement. En particulier, l'appareil photo et le GPS ne sont pas présents sur un PC.

Les fonctionnalités des PDA permettent à un utilisateur de sauvegarder ses données personnelles et professionnelles (contacts, e-mails, etc.), ce qui les rend d'autant plus vulnérables à une attaque. Le GPS permet de récupérer la position très précise du smartphone. Si le smartphone n'en est pas équipé, l'ID de la cellule GSM peut être une alternative pour localiser la zone approximative où se trouve le smartphone. Si l'appareil photo est utilisé par l'utilisateur, certaines photos peuvent être volées. Autrement, il peut être possible pour un attaquant de trouver des façons de prendre des photos sans que l'utilisateur du smartphone ne soit notifié. Les capacités du smartphone ouvrent la porte à un attaquant pour enregistrer des communications. Sinon, le microphone peut être activé afin d'écouter l'environnement proche du smartphone. Comme pour les PC, il peut être

1. Orange : http://couverture-reseau.orange.fr/france/netenmap.php, Bouygues Telecom : http://www.cartographie.bouyguestelecom.fr/eCouverture/eCouverture.aspx, SFR : http://assistance.sfr.fr/mobile_forfait/mobile/couverture-reseau/en-48-62267

envisageable d'utiliser un "keylogger" afin de voler les mots de passe de l'utilisateur.

État des lieux Des études ont été réalisées par Petr Matousek en 2007 sur les possibilités de rootkit pour Windows CE 5 pour masquer au cœur du système : processus, fichiers et clefs de registre [5].

Des portes dérobées pour Windows Mobile 6 existent actuellement. FlexiSpy, application commerciale, est dédiée à cette tâche. Elle permet d'expatrier des informations vers un serveur distant. Elle ne cherche cependant pas à se masquer complètement et elle est facilement détectable [6].

Les smartphones sont équipés d'une batterie. Détecter la présence d'un rootkit est facile s'il consomme trop la batterie et la vide rapidement. En effet, l'utilisateur final est directement impacté. Ceci fut le cas pour le soi-disant patch publié pour BlackBerry en juillet 2009 [7].

Apport Windows Mobile 6 repose sur Windows CE 5, les techniques de rootkit réalisées par Petr Matousek s'appliquent dans notre contexte. Windows Mobile ajoute cependant toute une couche dédiée aux applications mobiles : SMS, 3G, SD-card, GPS, etc. De plus, les mécanismes de sécurité implémentés par Microsoft et le programme Mobile2Market permettant aux développeurs d'applications de faire signer leurs applications sont à prendre en compte. De nombreux autres aspects à masquer à l'utilisateur sont donc à analyser.

La diversité des moyens de communication peut permettre au rootkit d'être accessible pour un attaquant. L'interception de SMS utilisée par FlexiSpy peut être détectée, car le smartphone s'allume automatiquement s'il est en veille. Nous avons trouvé un moyen pour que cela ne se produise plus. De plus, nous avons fait en sorte que le rootkit utilise n'importe quel moyen de communication accessible pour expatrier les informations : 3G, Wi-Fi, ActiveSync.

La gestion de la batterie est un aspect important de notre étude. Nous avons considéré le cas où le rootkit ne fait qu'utiliser des connexions existantes afin de réduire l'impact qu'il peut avoir sur le fonctionnement général du smartphone.

La diversité des services et la quantité de mémoire disponible nous ont obligés à nous adapter au monde de l'embarqué. Enfin,

nous avons implémenté une technique pour récupérer le code PIN de la carte SIM applicable à n'importe quel système mobile.

2 Aspects techniques de Windows Mobile 6

2.1 Historique

Windows Mobile, basé sur Windows CE, est le système d'exploitation de Microsoft adapté pour les smartphones. Windows CE est un système d'exploitation modulaire utilisé dans divers domaines : automobile, téléphonie, etc. Pour la version mobile, des modules dédiés à la téléphonie ou à l'appareil photo ont notamment été choisis. Depuis sa première commercialisation en 1996, Windows CE a subi plusieurs évolutions. La version Windows CE 5.x est présente depuis 2004. Il ne faut donc pas confondre la plateforme Windows Mobile et le coeur du système Windows CE [8]. Les systèmes Windows Mobile 5.x et 6.x développés pour les smartphones sont basés sur des versions modifiées de Windows CE 5. Ainsi, pour comprendre le fonctionnement interne de Windows Mobile 6, il suffit de comprendre le fonctionnement interne de Windows CE 5.

2.2 L'espace d'adressage virtuel

Un point important de Windows CE est son support pour de nombreuses APIs Win32, permettant aux développeurs de porter des applications du monde des PC vers le monde des systèmes embarqués. Cependant, des problèmes peuvent apparaître si de mauvaises méthodes de programmation sont utilisées ou bien si les mécanismes internes de Windows CE ne sont pas pris en considération. Cela est dû au fait que la gestion de la mémoire est différente entre les versions PC et mobile. Depuis Windows CE 3.0, la gestion de la mémoire a évolué de manière conséquente, afin de corriger les problèmes et limites que le système avait. Cependant, l'agencement général de la mémoire n'a pas changé jusqu'aux dernières versions de Windows CE 5.x [9].

Windows CE utilise un espace d'adressage virtuel de 4 Go. Les 2 Go supérieurs sont réservés au noyau et les 2 Go inférieurs sont pour les applications utilisateur. Les différences avec les versions Windows

PC commencent ici. Les 2 Go de la partie utilisateur sont divisés en 64 slots de 32 Mo chacun. Ils sont numérotés de 0 à 63. Chaque application s'exécutant se voit attribuer un numéro de slot dans l'intervalle [2-32]. Cela signifie que les applications s'exécutant sur le terminal se partagent l'espace d'adressage virtuel. Notons que le slot 97, qui se situe dans l'espace noyau, est le slot utilisé par le noyau (*nk.exe*). La figure 1 illustre cela.

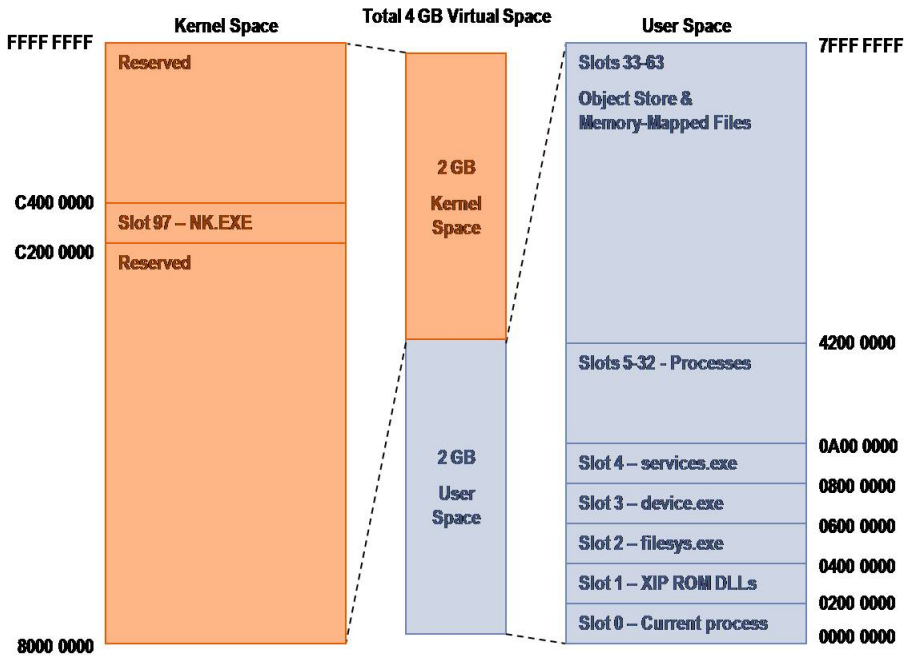

Figure 1. L'espace virtuel de Windows Mobile 6

Le slot 1 est utilisé pour les DLLs XIP (eXecute In Place). Ces DLLs sont des modules qui sont dans la flash du smartphone et ne sont pas copiés en RAM lorsqu'ils sont accédés. Ils sont directement mappés dans le slot 1 de l'espace d'adressage virtuel. Enfin, le slot 0 est utilisé pour le processus courant. Quand un thread appartenant à un processus donné est sur le point de s'exécuter, les 32 Mo du processus sont mappés au slot 0. Par conséquent, lorsqu'un thread s'exécute, toutes les informations du processus correspondant (code,

données, etc.) sont accessibles au slot 0 ainsi qu'au niveau du slot dans l'intervalle [2,32] où le processus réside (voir figure 2).

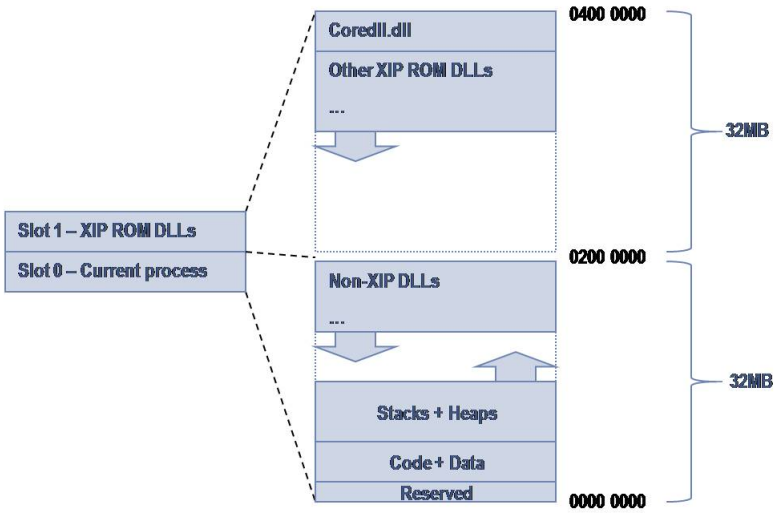

FIGURE 2. Slots 0 et 1

En se référant au livre de John Murray's Inside Microsoft Windows CE (1998), Dmitri Leman explique dans [10] les choix de Windows CE 3.0 et 4.0. Le slot 0 sert à optimiser le chargement des DLLs. Les sections de code et de données RO (Read Only) d'une DLL chargée par plusieurs processus sont mappées aux mêmes adresses physiques. Les sections de données W (Write) sont également au même endroit dans chaque slot de chaque processus, mais correspondent à des adresses physiques différentes.

Puisque le code a généralement des pointeurs embarqués sur les données, et qu'une seule copie du code est permise, ces pointeurs ne peuvent pas être modifiés lors d'un changement de processus, mais chaque processus doit pouvoir accéder à ses propres données. Ceci est réalisé en dirigeant tous les pointeurs vers le slot 0, où le processus courant est mappé. Par conséquent, quand une DLL s'exécute, elle accède automatiquement aux données privées du processus courant. Les pointeurs vers le slot 0 comme ceux-là sont dits *unmapped*. Nous pouvons accéder aux mêmes données (et code) en

mappant le pointeur au slot permanent du processus concerné en utilisant la fonction `MapPtrToProcess`. Cela est nécessaire lorsqu'un pointeur est passé d'un processus à un autre, par exemple comme argument d'une API ou d'un message Windows. Nous pouvons accéder aux adresses *mapped* depuis n'importe quel thread (dans n'importe quel processus) qui a la permission de le faire. La permission est un masque de 32 bits ou chaque bit correspond à chaque numéro de slot devant être accédé. Notons que lorsqu'un appel système est réalisé, dans le cas où certains arguments sont des pointeurs, le système les mappe automatiquement du slot 0 au slot du processus appelant (Caller).

Sous Windows CE 5.x, deux fonctions sont particulièrement intéressantes. `SetKMode` permet de passer en mode noyau et donc d'accéder à l'espace des 2 Go supérieurs. `SetProcPermissions` permet à un processus de modifier ses propres permissions afin de pouvoir accéder aux autres slots des autres processus qui lui sont restreints par défaut. Ces APIs rendent le système très permissif puisqu'un logiciel malveillant peut alors accéder à l'ensemble des fonctionnalités de Windows CE.

2.3 Le chargement des DLLs

Le chargement des DLLs est également partagé entre les processus. Comme expliqué précédemment, les DLLs XIP sont mappées dans le slot 1. Une DLL donnée est toujours mappée à la même adresse dans ce slot pour tous les processus. Ainsi, si un processus ne charge pas une DLL B, il devra garder de la place dans ce slot au cas où il la chargerait plus tard (pour que la règle s'applique).

Jusqu'à maintenant, nous avons discuté uniquement des DLLs XIP, mais des DLLs qui ne sont pas XIP peuvent également être utilisées (par exemple, des DLLs installées par une application). Ces DLLs sont chargées à l'intérieur du slot de 32 Mo du processus et la même règle s'applique i.e. qu'elles sont chargées à la même adresse quand elles sont mappées au slot 0 quelque soit le processus considéré.

La figure 3 montre trois processus s'exécutant sous Windows Mobile. Le processus 1 a démarré en premier et a chargé la DLL A et la DLL B. Ensuite, le processus 2 a démarré et a chargé la DLL A et

la DLL C. Au cas où il aurait besoin de charger la DLL B, l'OS lui garde de la place entre les deux DLLs A et C. Ensuite, le processus 3 démarre et charge la DLL A, la DLL B et la DLL D. Comme on peut le voir, si l'un des processus tente de charger une DLL supplémentaire trop grande, cela peut causer un problème puisqu'elle sera chargée sous la DLL D. Chaque processus possède son propre code, ses propres données, piles et tas. Il est donc possible qu'un processus manque de place pour charger une autre DLL dans son slot de 32 Mo. Pour plus d'information sur le chargement des DLLs, le lecteur pourra se référer aux très bons articles [11] et [12].

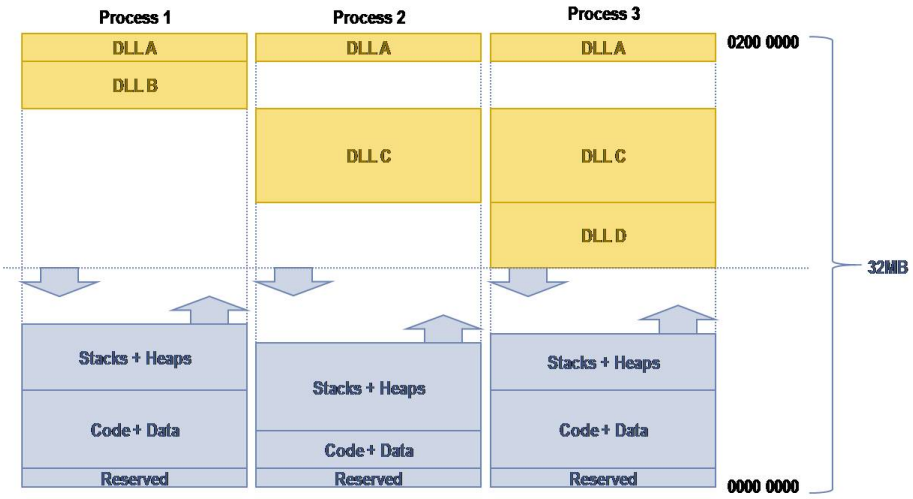

Figure 3. Le problème du chargement des DLLs

2.4 Appels systèmes

Windows CE implémente les modes utilisateur et noyau. Cependant, une autre différence entre les systèmes basés sur Windows PC et les systèmes basés sur Windows CE est la façon de gérer les appels systèmes. Windows CE utilise un saut vers une adresse invalide spécialement fabriquée pour réaliser un appel système. Cela cause une exception *prefetch abort* et la fonction du noyau gérant cette

exception détectera qu'il s'agit d'une adresse correspondant à un appel système et sera donc en mesure de gérer l'appel. Plus précisément, l'adresse invalide peut être décomposée en un APISetID et un MethodID précisant l'index au sein de l'APISetID. Cela peut être résumé par la formule suivante :

```
0xF0010000 - ((ApiSetID)«8 | (MethodID))*4
```

Le noyau (*nk.exe*) exécute un appel système en passant le thread en mode noyau, ajustant les paramètres de l'appel et exécutant le thread au sein du processus dédié à cet appel système. Ce processus est appelé PSL (Protected Server Library) et s'exécute dans l'espace utilisateur. Ensuite, la main est retournée au noyau qui restaure les permissions au thread, qui finalement retourne au niveau de l'application appelante. Chaque APISetID se voit attribuer un PSL pour gérer toutes les méthodes spécifiques à cet API Set. Par exemple, la fonction `FindFirstFileW` se voit attribuer l'adresse `0xF000AFE0`, qui correspond au couple [APISetID, Method ID] : [20, 8]. Cet APISetID est géré par le processus *filesys.exe*.

Il existe deux types d'APIs :

- Les APIs "implicites" sont celles qui ne sont pas basées sur un `HANDLE` (Ex : `FindFirstFileW`)
- Les APIs "explicites" sont celles qui sont explicitement basées sur un `HANDLE` (Ex : `FindNextFileW`)

2.5 Les 2 processeurs

Les smartphones, quelque soit leur constructeur, ont souvent l'architecture suivante. Ils possèdent un processeur applicatif (ou CPU) et un modem. Le CPU gère le système d'exploitation (dans notre cas : Windows CE). Le modem exécute un système d'exploitation temps réel qui gère toutes les communications avec le réseau cellulaire. Cette puce dédiée à la partie téléphonie est souvent appelée "baseband". En particulier, elle gère l'établissement d'appels téléphoniques ou la réception/l'envoi de SMS.

La communication entre le CPU et le modem se fait par une communication série. Le protocole utilisé est basé sur des commandes AT.

3 Implémentation

3.1 Architecture générale

Cette partie détaille la création d'un PoC (Proof of Concept) d'un rootkit installé sur un terminal utilisant Windows Mobile 6 et expatriant des informations sans que l'utilisateur légitime du smartphone en soit notifié. Nous détaillerons les choix techniques réalisés et les problèmes rencontrés.

Choix techniques Sous Windows Mobile 6, la zone mémoire disponible pour chaque processus est de 32 Mo. Elle regroupe l'image de l'exécutable, les différents threads exécutés dans le contexte du processus, ainsi que les DLLs chargées par le processus [13]. Éventuellement, dans les dernières versions de Windows Mobile 6.X, une zone supplémentaire peut être dédiée au chargement des DLLs [14]. De plus, ce chargement des DLLs est « partagé » entre les différents processus. En effet, une DLL chargée à une adresse par un processus force un autre processus à également garder de la place au même endroit, au cas où ils auraient besoin de charger cette DLL [11].

Ainsi, le chargement des DLLs peut influencer sur l'état général du terminal. Il a donc été préféré d'implémenter le module principal du rootkit sous forme d'un unique exécutable (.exe). À cela, nous ajouterons deux DLLs injectées pour nos besoins dans des processus du système. La taille de ces DLLs sera donc minimisée.

De plus, Windows Mobile est limité à 32 processus, cependant, il n'y a pas de limite (théorique) quant au nombre de threads possibles. Ainsi, nous avons choisi d'avoir un environnement multithreadé afin de partager les tâches à réaliser. En effet, nous souhaitons pouvoir exécuter les ordres demandés par l'attaquant et qu'en même temps une communication ait lieu avec le serveur distant. De même, nous souhaitons que l'exécution d'un ordre ne bloque pas la réception d'un SMS.

Cela nous amène également à discuter de la batterie du smartphone. Afin qu'elle ne s'épuise pas trop rapidement, une bonne stratégie est d'utiliser au maximum les notifications plutôt que de faire du polling. Ainsi, nous préférerons utiliser les APIs de notification offertes par Windows Mobile 6.

Schéma global Nous avons défini l'architecture générale du rootkit de la façon suivante. Le processus principal est un exécutable : *Engine.exe*. À cela s'ajoutent deux DLLs : *HookFS.dll* et *HookATCmds.dll* (cf figure 4).

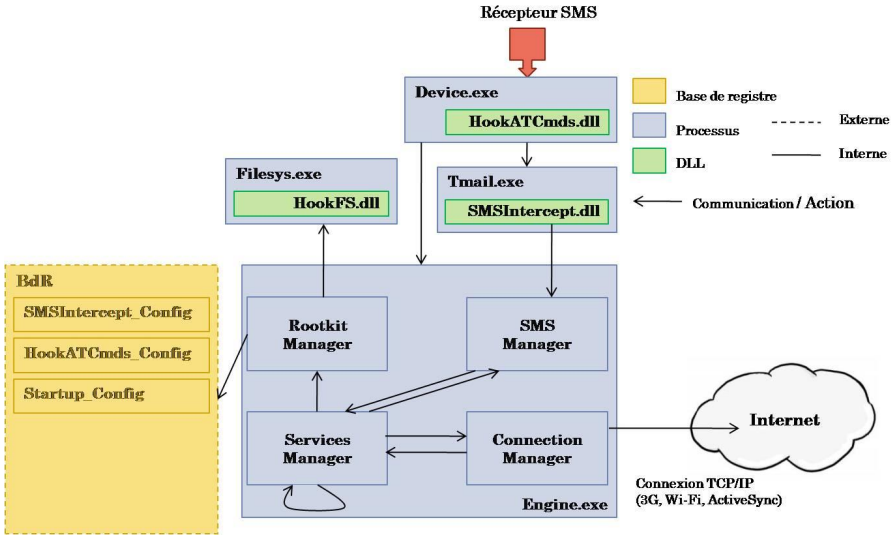

Figure 4. Architecture générale du rootkit

Le processus principal est constitué de quatre composants : *Rootkit Manager*, *SMS Manager*, *Services Manager*, *Connection Manager*. Le *Rootkit Manager* s'occupe de masquer à l'utilisateur la présence du rootkit afin qu'il soit le plus furtif possible. Le *Services Manager* s'occupe de gérer et d'exécuter les ordres de l'attaquant. Le *Connection Manager* fait en sorte de profiter des connexions TCP/IP existantes pour expatrier les informations (Wi-Fi, GPRS, 3G, ActiveSync). Enfin, le *SMS Manager* permet de prendre en compte le moyen de communication additionnel des messages SMS.

HookATCmds.dll est chargée dans le processus gérant la communication entre le modem et le CPU afin d'intercepter les SMS. Un filtrage est effectué afin de savoir si le SMS est dédié à notre rootkit ou bien s'il doit être distribué à la boîte de réception (Inbox) de l'utilisateur du smartphone.

HookFS.dll est injectée dans le processus *filesys.exe* afin de hooker certaines APIs. Ceci permet de masquer les fichiers et clefs de registre du rootkit à l'utilisateur final.

3.2 Injection

Cette section détaille les méthodes existantes pour injecter un rootkit sur un smartphone Windows Mobile 6.

Accès au smartphone La première méthode consiste à injecter le rootkit en ayant un accès physique au smartphone et à l'installer manuellement. Elle peut se faire en téléchargeant l'application depuis un serveur sur Internet ou bien en insérant une SD-card dans le smartphone.

FlexiSPY est une application vendue depuis 2006 par une entreprise commerciale chargée d'enregistrer les informations du smartphone et de les expatrier sur un serveur distant de la société. Les systèmes d'exploitation supportés sont : Symbian, BlackBerry, Windows Mobile et iPhone [15]. L'installation de l'application malveillante nécessite un accès physique au smartphone.

Exploitation à distance d'une vulnérabilité Il n'y a actuellement pas de méthodes publiques de rootkit s'injectant en utilisant des vulnérabilités sous Windows Mobile 6. Cela peut sans doute s'expliquer par le fait que très peu de vulnérabilités exploitables à distance ont été dévoilées publiquement pour ce système.

Le message WAP Push Une méthode alternative serait d'utiliser un message SMS Wap Push. Ces messages sont normalement envoyés par l'opérateur téléphonique ou l'administrateur du smartphone afin de configurer le smartphone à distance.

Début juillet 2009, certaines personnes possédant un BlackBerry et abonnées à l'opérateur Etisalat aux Émirats Arabes Unis ont reçu un message SMS indiquant qu'un patch était disponible pour leur smartphone Blackberry dans le but d'améliorer les performances du service Etisalat. Un lien hypertexte était disponible dans le message

SMS et l'utilisateur était invité à le télécharger [16]. Cependant, certaines personnes ont regardé de plus près l'application et ont découvert qu'elle s'appelait *Interceptor*. De plus, le code utilisé indiquait clairement que l'application tentait de transmettre une copie des e-mails envoyés par l'utilisateur à un serveur distant appartenant à Etisalat [17].

Le vecteur d'attaque était donc un message Wap Push avec un lien Web pour télécharger le spyware. Des techniques de social engineering invitant gracieusement l'utilisateur à l'installer ont donc été employées.

Injection par bootloader Avant de démarrer le système d'exploitation Windows CE, un bootloader (nommé SPL) est chargé d'initialiser le matériel. L'exploitation d'une vulnérabilité au niveau du bootloader permettrait d'avoir un accès complet au smartphone indépendamment des protections mises en place par Windows CE. En particulier, la présence d'un code de protection (screenlock) serait outrepassée. Il suffirait alors de reconstruire le système de fichiers en mode bootloader et d'écrire le rootkit dans la mémoire flash. La communication avec le bootloader se fait la plupart du temps avec une liaison série (encapsulée dans l'USB), cela permettrait d'injecter l'exploit à partir d'un ordinateur, mais aussi à partir de systèmes moins encombrants et plus discrets comme les microcontrôleurs.

Notre contexte Nous supposerons qu'il peut être installé en ayant un accès au smartphone en cliquant dessus. Cela s'applique au cas où une personne malintentionnée connaît sa victime et peut donc avoir accès à son smartphone pendant quelques minutes. L'attaquant pourrait soit télécharger son application malveillante depuis Internet (3G, GPRS), soit insérer une SD-Card et l'installer sans que sa victime ne soit là.

3.3 Protection

Cette section détaille les techniques implémentées pour protéger le rootkit, le rendre persistant à un redémarrage et faire en sorte qu'il soit le plus furtif vis-à-vis de l'utilisateur.

Démarrage automatique Nous nous intéressons ici aux méthodes permettant à une application de persister à un redémarrage du terminal. Un article résumant les techniques existantes peut être trouvé dans [18].

Répertoire \windows\startup. La première méthode proposée par Windows Mobile 6 est de créer un raccourci vers son application dans le répertoire *\windows\startup*. Les applications de ce répertoire sont exécutées au démarrage du terminal. Il suffit d'appeler la fonction `SHCreateShortcut`[2] avec comme premier argument la chaîne de caractère *\windows\startup\<nom_raccourci>.lnk* et comme second argument le programme dont on veut créer un raccourci [19].

C'est cette méthode que nous avons implémentée car elle s'impose par sa simplicité.

Clef de registre HKLM\Init. Alternativement, nous pouvons utiliser la clef de registre *HKLM\Init*. En effet, lorsque Windows CE démarre, le noyau examine cette clef de registre afin de démarrer les applications nécessaires. Cette méthode est en particulier utilisée pour charger les processus du système d'exploitation (*gwes.exe*, *device.exe*, *services.exe*, etc.).

Pour chaque application, deux clefs de registre sont à définir : *LaunchXX* et de manière optionnelle *DependXX* où XX est à remplacer par un numéro. Ce numéro correspond à un numéro de séquence et indique l'ordre dans lequel les applications seront exécutées. De plus, la clef *DependXX* permet d'indiquer une éventuelle dépendance. Pour plus d'informations, le lecteur pourra se référer à l'article [20].

Masquer la présence d'applications non signées Par défaut, lors de l'exécution d'une application non signée, l'utilisateur est sollicité avant d'exécuter l'application. L'application ne sera effectivement exécutée que si l'utilisateur accepte le pop-up (cf figure 5).

Une application malveillante souhaitera donc faire en sorte qu'aucun pop-up ne s'affiche. La suite de cette section détaille les méthodes envisageables.

2. http://msdn.microsoft.com/en-us/library/aa453680.aspx

Figure 5. Pop-up dû au fait que l'application n'est pas signée

1ère possibilité. La première méthode consiste à désactiver la politique n°*4122* (*Unsigned Prompt Policy*). Cette politique de sécurité permet de spécifier si une application non signée doit ou non afficher un pop-up. Ceci se configure dans la base de registre [21]. Michael Becher, Felix C. Freiling et Boris Leider ont utilisé cette méthode en 2007 lors de leur évaluation sur les efforts pour créer un ver pour smartphone sous Windows Mobile 5 [22].

Par défaut, lors du chargement de *HookATCmds.dll* par *device.exe* ou de *HookFS.dll* par *filesys.exe*, un pop-up est affiché. En désactivant cette politique de sécurité, même si nous n'avons pas signé nos exécutables, aucun pop-up n'est affiché.

2e possibilité. La première méthode n'est pas satisfaisante, car toute application externe non signée ne générera aucun pop-up. Un utilisateur averti détectera alors facilement que la politique de sécurité a été modifiée et que son smartphone est compromis.

Une méthode alternative consiste à signer les binaires de son application avec son propre certificat et à installer le certificat dans le magasin prévu à cet effet sur le terminal. Ceci permet d'éviter de générer des pop-up pour tous les binaires de son application, sans réduire le niveau de sécurité global du terminal. C'est donc cette

méthode qui a été implémentée. Il est donc tout à fait possible pour n'importe quelle application d'installer ses propres certificats, ce qui n'est pas très satisfaisant en matière de sécurité.

Masquer les processus utilisés

1ère possibilité. Lors de notre étude, il est apparu que Windows Mobile 6 ne permet pas nativement à l'utilisateur de lister tous les processus s'exécutant sur le terminal. Par défaut, seuls les processus possédant une fenêtre (au sens de Windows) sont visibles. Un processus s'exécutant en arrière-plan ne sera pas listé par le gestionnaire des tâches (« Task Manager »). Par exemple, *filesys.exe* ou *services.exe* n'apparaissent pas. Cependant, une application listant les processus ajoutée manuellement par l'utilisateur peut très bien lister tous ces processus.

Le gestionnaire des tâches de Windows Mobile 6 s'accède par *Paramètres* > *Système* > *Gestionnaire des tâches*, puis *onglet : En cours*. Nous avons également utilisé l'application fournie dans le SDK de Windows Mobile 6 sous le nom de "TrayTaskList". Ce sample permet de lister tous les processus s'exécutant à un instant donné.

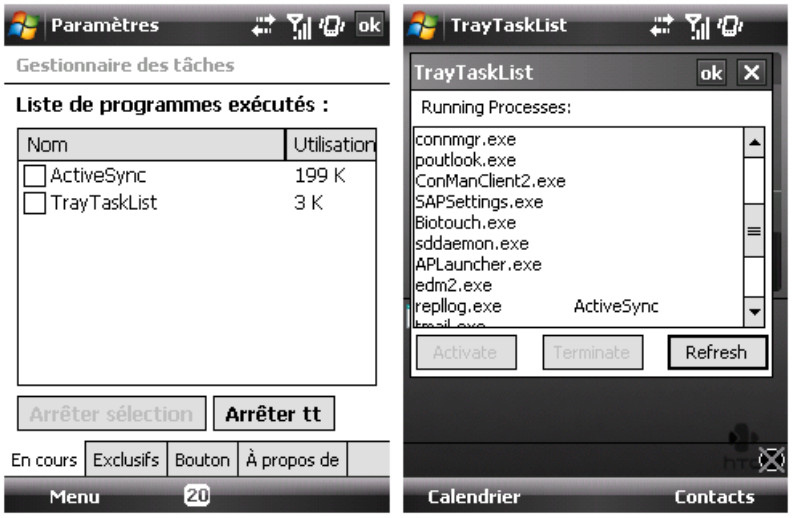

Figure 6. Gestionnaire des tâches vs processus réellement exécutés

Comme nous pouvons le voir figure 6, les processus démarrés sont bien plus nombreux que ce que nous montre le gestionnaire des tâches. De plus, l'application "TrayTaskList" nous indique le nom de la fenêtre associée si elle est présente, comme c'est le cas pour *repllog.exe* qui affiche "ActiveSync". Nous pouvons donc conclure que n'importe quelle application qui s'exécute en arrière-plan et n'a pas de fenêtre associée ne sera jamais affichée dans le gestionnaire des tâches et sera par conséquent masquée à l'utilisateur. Ceci représente un risque pour ce dernier.

2e possibilité. Petr Matousek s'est intéressé à masquer les processus dans [5]. Il a étudié la fonction `SC_THCreateSnapShot` et plus particulièrement `THGetProcs`.

Sous Windows Mobile 6, il n'y a pas de listes doublement chaînées pour gérer les processus. En effet, Windows Mobile 6 ne supporte que 32 processus. Ainsi, un tableau de slots est utilisé, chaque slot correspondant à un processus. La fonction `THGetProcs` parcourt ce tableau à la recherche de processus démarrés avant de continuer. Une des conditions pour considérer qu'un slot est utilisé par un processus est que `ProcStarted(P)` retourne "vrai", où `P` est un pointeur vers la structure `Processus` (cf listing 1.1).

```
#define VA_SECTION      25
#define ProcStarted(P) (((P)->dwVMBase>>VA_SECTION)==((DWORD)((
    P)->lpszProcName)>>VA_SECTION))
```

Listing 1.1. Définition de la macro ProcStarted

Le champ "`lpszProcName`" correspond au nom du processus. Il suffit donc de mettre à `NULL` ce pointeur pour qu'il ne soit plus énuméré par la fonction `SC_THCreateSnapShot` et donc invisible d'un listing des processus par les APIs de Windows Mobile. Ces structures étant gérées dans l'espace noyau, il suffit d'appeler la fonction `SetKMode` pour pouvoir les modifier. Cette méthode permet donc de ne plus apparaître même lors de l'utilisation d'une application tierce pour lister les processus.

Masquer les fichiers et clés de registre utilisés Comme sur un PC, l'utilisateur peut parcourir les répertoires et fichiers en utilisant un explorateur de fichiers (cf figure 7).

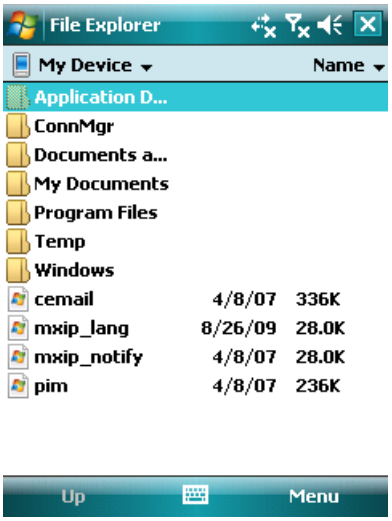

FIGURE 7. Explorateur de fichiers (File Explorer) sous WM6

Windows Mobile utilise la base de registre pour sauver les paramètres des programmes ou du système d'exploitation (ex. : les politiques de sécurité). Il n'y a pas de logiciel permettant de visualiser cette base de registre par défaut (comme *regedit* sous Windows pour PC) mais l'utilisateur peut installer sa propre application. Un logiciel comme PHM Registry Editor[3] permet de visualiser les éléments de la base de registre, directement sur le terminal. Il est donc possible de détecter la clef de registre utilisée pour persister à un redémarrage (cf figure 8). De même, l'enregistrement de l'interception des SMS détaillé par la suite peut être détecté dans la base de registre.

Hook d'APIs. Plusieurs travaux ont été réalisés sur les méthodes de hook d'APIs sous Windows Mobile 6. Trois prérequis sont nécessaires pour comprendre ces travaux :

- Sous Windows CE 5, un appel système est réalisé par un saut vers une adresse de la zone mémoire `0xF0000000` à `0xF0010000`. Cette zone mémoire est notée comme invalide, ce qui génère une "prefetch abort exception". Le gestionnaire d'exception implémenté sous Windows CE 5 récupère cette adresse et détecte

3. http ://www.phm.lu/Products/PocketPC/RegEdit/

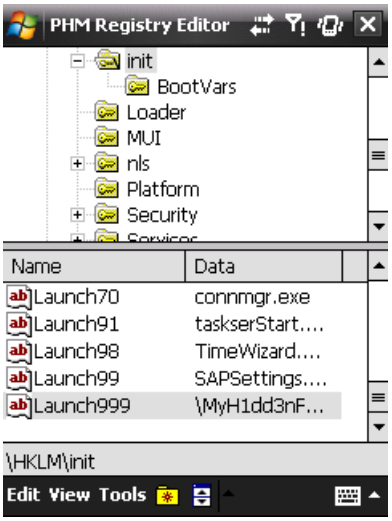

Figure 8. Configuration pour le démarrage automatique dans la base de registre

qu'il s'agit d'une adresse de cette zone mémoire pour la considérer comme un appel système. En interne, chaque adresse code un couple (*ApiSet*, *MethodID*). Un *ApiSet* est une liste de méthodes, le *MethodID* indique l'index dans cette liste.

- Sous Windows CE 5, la majorité des appels systèmes sont gérés par des processus en mode utilisateur, nommés PSL ("Protected Server Library"), par exemple : *device.exe*, *filesys.exe*, etc.
- Il existe deux types d'APIs sous Windows CE 5, les APIs "implicites" et les APIs "explicites". Le premier type concerne les APIs n'étant pas basées explicitement sur un `HANDLE` tandis que le second type concerne les APIs "explicitement" basées sur un `HANDLE`. Le gestionnaire d'exception saute alors vers une adresse dédiée du SPL. L'adresse vers laquelle ce saut a lieu est stockée dans des structures gérées par le noyau :
  - Pour les APIs "implicites", un tableau global `SystemApiSets[]` est utilisé ;
  - Pour les APIs "explicites", `((HDATA*) HANDLE)->pci` est utilisé.

La modification de ces adresses permet de détourner le flot d'exécution normal des programmes faisant appel à ces APIs. Comme ces structures se trouvent dans l'espace noyau, il suffit d'appeler la fonction `SetKMode` pour pouvoir les modifier.

Dmitri Leman publia dès 2003 l'outil *CeApiSpy* (Windows CE API Spy) ainsi que son code source permettant de déboguer des applications sous Windows CE. Cet outil utilise des techniques de hook au niveau du noyau afin d'intercepter les appels de fonctions et affiche les paramètres passés à chaque appel dans une fenêtre sur le terminal (cf figure 9). Notons que seules les APIs "implicites" sont supportées. Par défaut, il intercepte les APIs : `CreateProcess`, `CreateFile` et `LoadLibrary`. Cet outil était pour Windows CE 3.0 et 4.0 mais une version mise à jour est sortie en 2005 supportant Windows CE 5.0 et Windows Mobile 2005 [10].

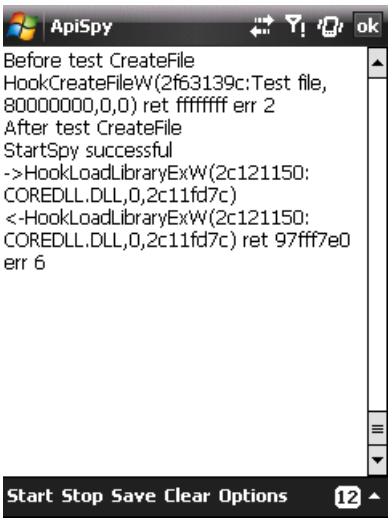

Figure 9. Application *CeApiSpy* sur un terminal WM6

Petr Matousek a publié ses travaux sur les rootkits pour Windows CE 5 en 2007, détaillant comment hooker les fonctions "implicites" et "explicites" du noyau sous Windows CE 5. L'auteur s'est intéressé aux possibilités de cacher des fichiers et clefs de registres. Dans le cas des fichiers, il s'agit de hooker les fonctions `FindFirstFileW` et

`FindNextFileW`, correspondant aux deux cas d'APIs implicites et explicites vus ci-dessus. Pour les clefs de registres, il s'agit de hooker `RegEnumValue` et `RegEnumKeyEx`.

Compte tenu du fait que Windows Mobile 6 se base sur Windows CE 5, ces recherches s'appliquent également à Windows Mobile 6. Toutes ces fonctions sont gérées par le PSL *filesys.exe*. Ceci explique pourquoi nous avons choisi d'injecter une DLL (*HookFS.dll*) dans ce processus. Nous effectuons le hook en deux étapes. Tout d'abord, nous injectons notre DLL dans *filesys.exe*. Ensuite, nous faisons appel à une fonction exportée de la DLL afin d'installer le hook. Pour exécuter une fonction dans le contexte du processus distant, la fonction non documentée `PerformCallBack4` peut être utilisée.

Masquer l'installation du CAB Sous Windows Mobile 6, l'utilisateur peut visualiser la liste des applications installées en accédant au menu *Paramètres* > *Système* > *Suppr. de progr.*. La gestion des applications installées se fait dans la base de registre au niveau de *[HKLM\Security\AppInstall]*. Une clef est créée dans cette arborescence pour chaque application installée.

1ère possibilité. La première méthode a été trouvée en effectuant du reverse engineering sur l'application "Airscanner Mobile Firewall". Cette application possède une option permettant de faire disparaître l'application de la liste (cf figure 10). Elle consiste à spécifier la valeur "Role" à 0.

2e possibilité. La première méthode n'est pas satisfaisante pour que le rootkit soit complètement furtif puisqu'il suffit d'utiliser un visualisateur de base de registre pour visualiser d'éventuelles applications tentant d'être "furtives". Une seconde méthode a été trouvée, et concerne directement la configuration de Visual Studio. Il suffit de spécifier l'option *"NoUninstall"* dans un projet CAB (cf figure 11). En effet, ceci a pour effet de ne pas créer de clef dans *[HKLM\Security\AppInstall]*. Il n'y a donc aucune façon de détecter l'installation de l'application dans la base de registre.

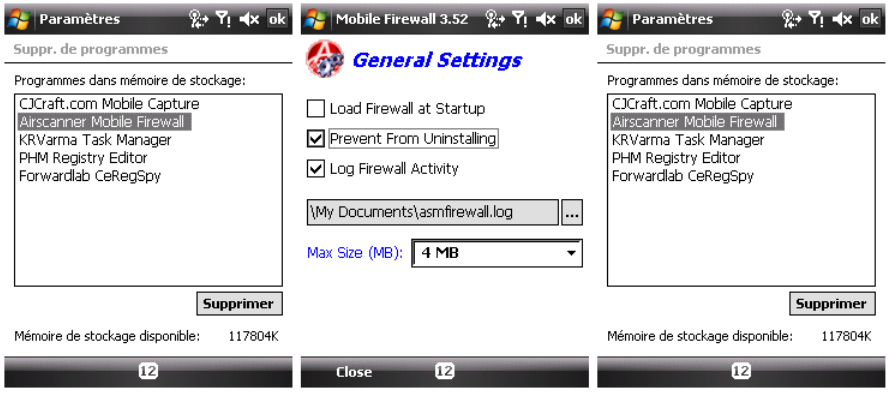

FIGURE 10. Airscanner Mobile Firewall (c)

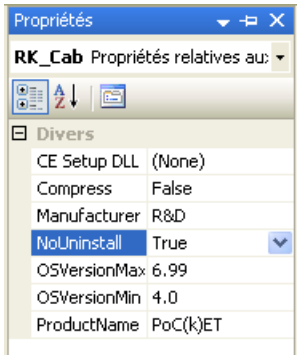

FIGURE 11. Option *NoUninstall*

3.4 Backdoor

Cette section s'intéresse aux moyens de communication qui permettent de commander à distance le smartphone et/ou récupérer des informations.

Les moyens de communication d'un smartphone sont très différents des moyens de communication offerts par un PC. De nos jours, les PC sont toujours connectés à Internet. Ils offrent souvent une large bande passante en sens montant et descendant avec un accès illimité. Ces ordinateurs utilisent des CPU de plusieurs GHz. Ils sont équipés de plusieurs gigaoctets de RAM et ont des centaines

voire milliers de gigaoctets d'espace disque. Par conséquent, des personnes malintentionnées peuvent utiliser facilement ces accès et leurs ressources. Au contraire, les terminaux embarqués sont encore limités en ressources. Aujourd'hui, un smartphone a typiquement un CPU cadencé à 400 MHz avec 256 Mo de mémoire flash pour la sauvegarde de données et 128 Mo de RAM (cf figure 12).

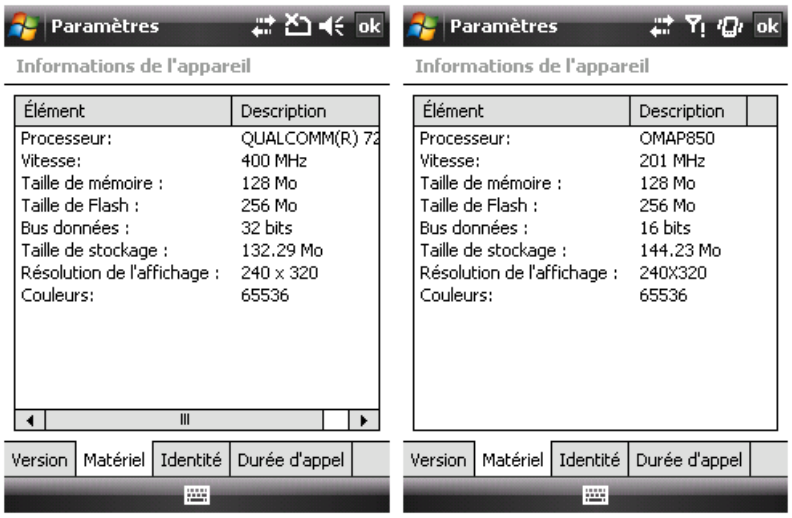

Figure 12. Matériel des smartphones HTC Kaiser et HTC Touch

Les smartphones offrent une variété de moyens de communication. Un smartphone offre nativement des possibilités de communication en utilisant : GSM, GPRS, 3G, Wi-Fi, Bluetooth, SMS, GPS, etc. Certains de ces moyens de communication (comme le GSM, SMS) coûtent un supplément à l'utilisateur. D'autres sont compris dans le forfait (ex. : 3G, GPRS, Edge). Enfin, il y en a également qui sont indépendants de l'opérateur (Bluetooth, Wi-Fi). Notons également que cela peut dépendre du contrat soumis par l'utilisateur (ex. : les SMS peuvent être illimités, compris dans un forfait global).

De plus, les smartphones évoluent dans un environnement hétérogène (réseaux d'opérateurs, obstacles physiques, etc.), donc la qualité de services (QoS) est très différente du monde des PCs. Par exemple, un smartphone peut ne pas être en mesure de recevoir des appels

téléphoniques dans certaines zones. La livraison d'un message SMS peut être retardée si l'opérateur télécom n'est pas en mesure de le gérer à l'instant t.

Communication TCP/IP Du point de vue de l'attaquant, un smartphone peut être attaqué en utilisant TCP/IP sur des connexions Wi-Fi, GPRS ou 3G. Des méthodes identiques à celles utilisées sur les PCs peuvent être adoptées pour contrôler à distance un éventuel rootkit présent sur smartphone. Ce sont ces méthodes qui sont utilisées par *FlexiSPY* pour expatrier des informations vers un serveur distant.

TCP/IP peut être considéré comme gratuit pour l'utilisateur s'il est utilisé sur du Wi-Fi. Dans le cas d'une connexion UMTS, il a longtemps été payant, mais devient de plus en plus souvent compris dans le forfait des utilisateurs (dans le cas de HTTP). Son défaut majeur est de n'être accessible que par intermittence.

Le smartphone est derrière un NAT. Un attaquant pourrait souhaiter se connecter à distance sur le terminal. Cependant, le smartphone peut ne pas être connecté directement à Internet comme le sont les ordinateurs d'aujourd'hui. Ceci dépend des pays et ne peut donc pas être généralisé. Ainsi, le smartphone n'a a priori pas d'adresse IP publique. Ceci est dû à un NAT implémenté dans le réseau de l'opérateur. Il serait donc impossible pour l'attaquant d'initier une communication avec le smartphone. Ainsi, nous avons choisi d'implémenter un serveur au niveau de l'attaquant. Nous supposerons qu'il est accessible par TCP/IP par un couple "nom_domaine/port" bien connu. C'est toujours la cible qui initie la connexion au serveur. Le serveur envoie alors la liste des informations souhaitées au smartphone. Le rootkit expatriera alors ultérieurement ces informations au serveur.

La gestion de la batterie. L'utilisation de la batterie doit également être prise en considération. Le spyware envoyé par l'opérateur Etisalat aux Émirats Arabes Unis en est un bon exemple. En effet, des suspicions se sont créées après que des utilisateurs se soient plaints que leur batterie s'épuisait trop rapidement depuis l'installation de ce soi-disant patch. D'après l'étude de Sheran A. Gunasekera dans

[17], cela était dû au fait que l'application effectuait des actions de polling afin de vérifier si un nouveau message de commande en provenance du serveur avait été reçu.

FIGURE 13. Pop-up d'établissement de connexion

Être connecté sans interruption vers un réseau de données (GPRS, UMTS, Edge) n'est pas nécessaire. Cela permet également d'économiser de la batterie. Lorsque le smartphone a besoin d'un accès "data", il établit une connexion avec le réseau de l'opérateur qui lui valide l'accès. Sous Windows Mobile, cela ouvre un pop-up indiquant que la connexion est en cours (cf figure 13). Dans un scénario où l'utilisateur du smartphone n'a pas demandé de connexion vers le réseau "data", un logiciel malveillant établissant une connexion peut donc révéler sa présence à l'utilisateur.

De plus, l'écran principal indique l'état actuel de la connexion comme le montre la figure 14. Même s'il ne voit pas l'établissement de la connexion, un utilisateur averti pourrait détecter la présence d'une utilisation malveillante du réseau par la simple icône indiquant qu'une connexion est établie.

Notons que l'établissement de connexions réalisé par une application malveillante sans que l'utilisateur n'utilise le réseau data im-

FIGURE 14. Affichage de l'état de connexion

pliquerait d'avoir une politique de fréquence/durée de connexion. En effet, il ne faudrait pas trop souvent se connecter afin de ne pas décharger la batterie trop rapidement. De plus, même si cela devient de moins en moins vrai, une connexion data peut être payante pour l'utilisateur. Un utilisateur pourrait donc détecter la présence d'une application malveillante en fin de mois en regardant sa facture détaillée.

Pour toutes ces raisons, nous pensons qu'un attaquant souhaitant rester furtif choisira de ne jamais établir de communication, mais de profiter de connexions existantes. Nous avons donc décidé d'implémenter cette stratégie. Pour ce faire, il n'était pas question d'implémenter une technique de polling qui irait vérifier l'état de la connexion régulièrement, sinon la batterie se serait également déchargée. La Connection Manager API [4] fournie par Windows Mobile répond à ses critères.

La fonction `ConnMgrRegisterForStatusChangeNotification` permet d'enregistrer un handle de fenêtre afin d'être notifié lors d'un changement d'état. Cependant, le problème est que la notification n'indique que le changement d'état : "connecté" ou "déconnecté".

4. http://msdn.microsoft.com/en-us/library/bb416435.aspx

Nous ne savons donc pas de quel type de connexion il s'agit : Wi-Fi, GPRS, 3G, ActiveSync ou même Bluetooth.

La fonction `ConnMgrEstablishConnection` permet d'établir une connexion en s'abstrayant du type de connexion réellement utilisé. Ainsi, il suffit d'indiquer quel réseau nous souhaitons atteindre pour que le Connection Manager détermine le meilleur chemin pour y accéder. Cependant, ici, nous ne souhaitons pas nous connecter, mais plutôt détecter la présence d'une connexion. Nous avons donc choisi de réimplémenter la recherche d'un chemin pour accéder au serveur distant en parcourant les structures `CONNMGR_CONNECTION_DETAILED_STATUS`. Nous utilisons la fonction `ConnMgrQueryDetailedStatus` [5] pour lister toutes les connexions configurées afin de déterminer si la connexion nouvellement effectuée nous intéresse. Pour cela, il suffit de regarder le champ "`dwConnectionStatus`" de la structure `CONNMGR_CONNECTION_DETAILED_STAT` en liant les réseaux grâce aux champs "guidSourceNet" et "guidDestNet".

Protocole. Pour communiquer en TCP/IP, nous avons défini deux types de requêtes qui peuvent être envoyées par le smartphone au serveur.

La première est une requête d'identification. Le serveur répond par la liste des ordres (services) qui lui sont destinés. La seconde correspond à une requête de dump des résultats pour un ordre donné. Ainsi, le smartphone envoie une requête de dump par résultat. Ceci à l'avantage de permettre de dumper seulement les résultats qui ont été récupérés à un instant donné.

La requête d'identification (1) précise l'identité du rootkit. La réponse (2) listant les ordres précise tout d'abord le nombre d'ordres. Pour chaque ordre, elle précise l'utilisateur le demandant et des données éventuelles. Dans le cas d'un ordre demandant un e-mail par exemple, les données correspondent à l'e-mail souhaité. La requête (3) permet de dumper les résultats d'un ordre donné (cf figure 15).

Compression. Afin d'optimiser l'envoi de données volumineuses OTA (Over The Air), nous avons décidé de compresser les données envoyées par le rootkit. En effet, celles-ci peuvent être conséquentes

5. http://msdn.microsoft.com/en-us/library/bb416284.aspx

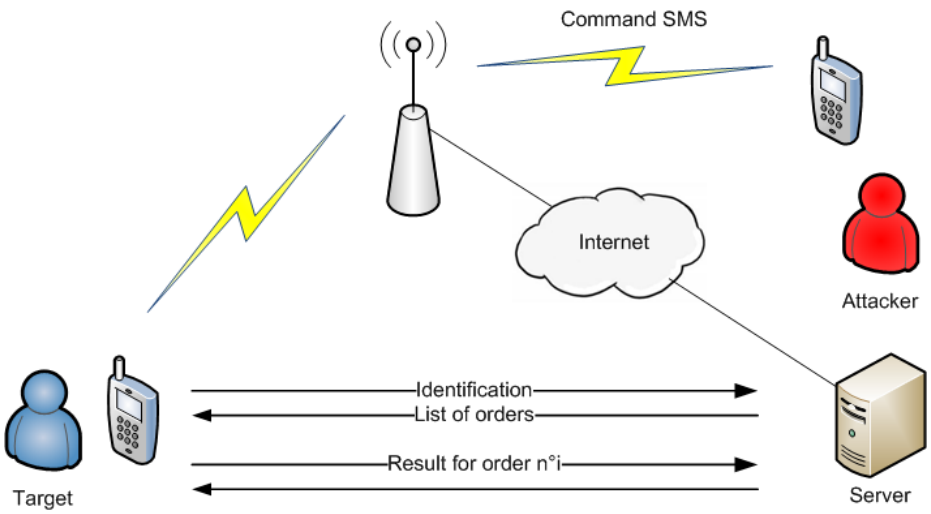

FIGURE 15. Protocole de communication entre l'attaquant et la cible

(contenu des SMS, e-mails, captures d'écran, etc.). Nous avons choisi d'utiliser la librairie *zlib* qui a été portée pour Windows CE [6].

Chiffrement des communications. Comme nous le détaillerons dans la section 3.5, de nombreuses données personnelles peuvent être volées sur un smartphone. L'attaquant peut souhaiter faire en sorte que ces données ne soient visibles que pour lui et qu'elles ne passent pas en clair sur le réseau de l'opérateur. Pour cela, Windows Mobile 6 fournit des APIs permettant d'établir des communications chiffrées. Nous avons cependant préféré utiliser l'API PolarSSL [7] (anciennement XySSL). Cette librairie dont les sources sont disponibles est particulièrement bien adaptée à l'environnement embarqué et se compile facilement pour Windows Mobile 6.

Un moyen alternatif ? Même si TCP/IP correspond à la meilleure façon d'expatrier des informations, un smartphone peut être dans une situation où il lui est impossible d'avoir une telle connexion (Edge, GPRS, UMTS, Wi-Fi ou connexion à un PC). Cependant, un

6. http://www.opennetcf.com/FreeSoftware/zlibCE/tabid/245/Default.aspx
7. http://www.polarssl.org/

attaquant peut souhaiter garder un contrôle sur un smartphone qu'il a infecté. L'envoi de SMS est le plus souvent payant, mais apparaît comme la meilleure solution quelque soit l'environnement. En effet, les messages SMS utilisent le réseau de service des opérateurs, qui est (presque) toujours disponible. Son défaut majeur est qu'il limite la quantité d'informations transmises à 140 octets.

1ère solution : utilisation des APIs offertes par Windows Mobile 6. Windows Mobile 6 offre nativement la possibilité d'intercepter des SMS avant qu'ils n'arrivent dans la boîte de réception (Inbox). La Messaging API (MAPI) [8] peut être utilisée pour cela. Deux fonctions doivent être définies au sein d'une DLL. `Initialize` [9] permet d'indiquer au composant de quel genre d'accès nous avons besoin. Dans le cas où l'on souhaite intercepter des messages SMS et qu'ils ne s'affichent pas dans la boîte de réception, il faut typiquement utiliser un accès en écriture. `ProcessMessage` [10] est appelée par l'OS lorsqu'un message est reçu. Le paramètre "`pMsgStore`" spécifie le magasin de messages. Le paramètre "`lpMsg`" est un pointeur vers l'ID du message intercepté. Il est donc possible de filtrer, supprimer ou déplacer les messages reçus (cf listing 1.2).

```
HRESULT Initialize (
  IMsgStore * pMsgStore,
  MRCACCESS * pmaDesired
);

HRESULT ProcessMessage (
  IMsgStore * pMsgStore,
  ULONG cbMsg,
  LPENTRYID lpMsg,
  ULONG cbDestFolder,
  LPENTRYID lpDestFolder,
  ULONG * pulEventType,
  MRCHANDLED * pHandled
);
```

Listing 1.2. Prototype des fonctions `Initialize` et `ProcessMessage`

8. http://msdn.microsoft.com/en-us/library/bb446125.aspx
9. http://msdn.microsoft.com/en-us/library/bb445992.aspx
10. http://msdn.microsoft.com/en-us/library/bb446199.aspx

Deux clefs de registre doivent être configurées (cf tableau 1) sur le terminal pour que *tmail.exe* charge notre DLL et que l'interception soit prise en compte.

Table 1. Clefs de registre à définir pour intercepter des messages SMS

Enregistrement standard COM	`HKEY_CLASSES_ROOT\CLSID\<clsid>`
Boîte de réception MAPI	`HKEY_LOCAL_MACHINE\Software\Microsoft\Inbox\Svc\SMS\Rules\<clsid>`
`<clsid>` représente le GUID (class ID) de l'objet COM.	

Notons que cette méthode est utilisée par *FlexiSPY* [6].

Il est nécessaire que le processus *tmail.exe* ne soit pas déjà lancé pour que les clefs de registre modifiées soient prises en compte. Ceci peut expliquer pourquoi FlexiSPY demande un redémarrage du terminal après installation. Ceci est par exemple le cas si l'utilisateur du smartphone a exécuté l'application "Messaging" afin de lire ses SMS (cf figure 16), depuis le démarrage de son smartphone.

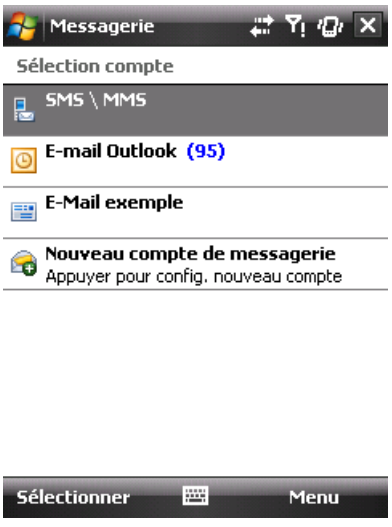

Figure 16. Application Messaging

Une solution a été trouvée pour résoudre ce problème et éviter de redémarrer le smartphone. En effet, il suffit de tuer le processus *tmail.exe* avant de mettre à jour les clefs de registre. Lors du redémarrage de *tmail.exe*, les clefs de registre seront vérifiées et prises en compte. La méthode utilisée consiste à parcourir la liste des processus grâce aux fonctions de la ToolHelp API [11] afin de récupérer l'ID du processus. Ensuite, il suffit d'utiliser la fonction `OpenProcess` afin de récupérer un `HANDLE` associé. Enfin, la fonction `TerminateProcess` permet de terminer le processus.

Lors de la réception d'un SMS de commande, ce dernier est intercepté et n'est pas présent dans la boîte de réception. Nous avons cependant mis en évidence un moyen de détection. En effet, si le téléphone est en mode veille, ce dernier s'allume automatiquement. Ceci permet à un utilisateur de détecter la présence du rootkit.

2e solution : interception des commandes AT. Comme détaillé en section 2.5, les smartphones possèdent deux processeurs, le CPU gérant le système d'exploitation et le baseband gérant les communications GSM. La communication entre ces deux processeurs se fait par des commandes AT. Sous Windows Mobile, un driver est chargé sous forme d'une DLL par le processus *device.exe* et permet la communication avec le baseband. Willem Jan Hengeveld a implémenté un driver permettant de hooker cette communication [23]. Il avait été repris en 2009 par Charlie Miller et Collin Mulliner pour réaliser du fuzzing. Ils ont injecté des SMS directement au niveau de ce driver permettant de simuler la réception de SMS en provenance du baseband [24]. Un exemple de SMS reçu est le suivant :

```
+CMT: ,44\r\n
079133161111111F1040B913326222222F2000001206111
3205401CC16030180C0683C16030180C0683C16030180C0683C1603008\r\n
```

Pour en comprendre le contenu, celui-ci peut être décodé en utilisant par exemple le logiciel PDUSpy [12](cf listing 1.3).

```
    PDU LENGTH IS 52 BYTES
    ADDRESS OF DELIVERING SMSC
        NUMBER IS : +33611111111
```

11. http://msdn.microsoft.com/en-us/library/aa914815.aspx
12. http://www.nobbi.com/pduspy.html

```
        TYPE OF NR. :   International
        NPI :   ISDN/Telephone (E.164/163)

    MESSAGE HEADER FLAGS
        MESSAGE TYPE :  SMS DELIVER
        MSGS WAITING IN SC :    NO
        SEND STATUS REPORT :    NO
        USER DATA HEADER :  NO UDH
        REPLY PATH :    NO

    ORIGINATING ADDRESS
        NUMBER IS : +33622222222
        TYPE OF NR. :   International
        NPI :   ISDN/Telephone (E.164/163)

    PROTOCOL IDENTIFIER (0x00)
        MESSAGE ENTITIES :  SME-to-SME
        PROTOCOL USED : Implicit / SC-specific

    DATA CODING SCHEME  (0x00)
        AUTO-DELETION : OFF
        COMPRESSION :   OFF
        MESSAGE CLASS : NONE
        ALPHABET USED : 7bit default

    SMSC TIMESTAMP :    16/02/10 11:23 GMT+1,00

    USER DATA PART OF SM
        USER DATA LENGTH :  28 septets
        USER DATA (TEXT) :  AAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Listing 1.3. Analyse du SMS reçu avec PDUSpy

Nous avons réutilisé ce driver pour intercepter les SMS directement au niveau des commandes AT. Il a donc été nécessaire d'implémenter un décodeur dans ce driver pour analyser les SMS reçus. Lors de la réception d'un SMS, le driver vérifie si un motif donné est présent. Si ce motif est présent, il intercepte le SMS et l'envoie à notre rootkit. Si le smartphone est en veille, il ne s'allume pas automatiquement, ce qui le rend totalement invisible pour l'utilisateur. Notre rootkit reçoit donc les ordres en laissant le smartphone en veille. Ceci permet d'économiser la batterie du smartphone tout en le rendant accessible pour l'attaquant.

L'inconvénient de cette méthode est qu'il est impossible de tuer le processus *device.exe* sans faire crasher le terminal. Ainsi, cette méthode nécessite un redémarrage du smartphone.

Protocole et encodage. Afin de pouvoir envoyer le même genre de commandes par SMS qu'en utilisant TCP/IP, un protocole simi-

laire a été défini. Ce protocole utilise un format binaire. Pour éviter tout problème d'encodage du SMS dû au transport sur le réseau de l'opérateur, nous avons décidé d'encoder le contenu du SMS au format base 64. Ceci permet de n'utiliser que des caractères ASCII.

3.5 Services

Le comportement des gens envers leur téléphone portable a évolué. En effet, en évoluant vers les smartphones, il est devenu l'appareil que les gens ont toujours sur eux. Ainsi, il sert de répertoire de contacts, de lecteur d'e-mails ou SMS, mais également de bloc-notes pour y noter ses "pense-bêtes" ou rendez-vous. Enfin, l'utilisateur y stocke ce qu'il utilise tous les jours : ses applications préférées, la musique qu'il écoute, les photos qu'il prend. Tout ceci correspond aux données personnelles et professionnelles qui sont chères à l'utilisateur et aux entreprises. Tous les services offerts par un smartphone se veulent satisfaire l'utilisateur, mais ils peuvent devenir un réel cauchemar s'ils sont utilisés de manière malveillante par un attaquant sans que l'utilisateur ne le sache. Cette section évalue les possibilités pour un attaquant d'accéder également à ces services.

Types de services La quantité de mémoire étant relativement limitée, certains résultats seront expatriés au moment de leur récupération (e-mail, SMS). D'autres peuvent nécessiter d'être exécutés même si aucune connectivité TCP/IP n'est disponible (coordonnées GPS, ID de la cellule GSM). Nous distinguerons trois types de services :

- Les services qui s'exécutent lors de la communication avec le serveur et sont dumpés au fur et à mesure de leur récupération ;
- Les services qui s'exécutent dès leur réception et dont le résultat est sauvegardé dans un fichier ;
- Les services qui sont activés/désactivés par un attaquant suivant le besoin.

Il apparaît que la majorité des services sont disponibles en utilisant les APIs de Windows Mobile 6 [1]. Un utilisateur familiarisé avec l'environnement Windows n'aura pas de mal à utiliser les APIs de Windows Mobile.

Services précalculés. Certains services ont besoin d'être exécutés en arrière-plan afin de récupérer les informations lorsqu'elles sont acces-

sibles. Typiquement, les coordonnées GPS du terminal nécessitent un certain temps avant d'être disponibles. Ceci peut durer plusieurs minutes. C'est le module "Services Manager" qui se chargera de la bonne exécution de ces services.

Services non précalculés. Compte tenu des ressources relativement limitées du smartphone, il a été choisi de ne pas dupliquer dans des fichiers toutes les informations récupérées par le rootkit avant de pouvoir effectivement dumper les résultats vers le serveur distant. L'exemple typique concerne les e-mails qui peuvent contenir plusieurs Ko voire Mo de données. Ainsi, c'est le "Connection Manager" qui se chargera de l'exécution de ces services lors de la phase de connexion avec le serveur distant.

Services en arrière-plan. Le dernier type de service concerne la récupération régulière d'informations. Par exemple, la localisation est d'autant plus intéressante qu'elle est récupérée à différents moments. Ce type de service est activé par une commande qui indique la fréquence d'exécution et reste actif jusqu'à ce que l'attaquant souhaite l'arrêter. Dans un souci de ne pas vider la batterie du smartphone et d'être détecté, il faudra cependant que cette fréquence ne soit pas trop importante.

Informations PIM Nous appelons classiquement informations PIM [13] certaines informations personnelles comme les contacts, e-mails, SMS, etc.

Contacts. La POOM API [14] permet d'accéder aux contacts du smartphone. Cette API est de type COM. Pour chaque contact, nous récupérons nom, prénom, adresse(s) e-mail, entreprise, numéro(s) de téléphone, adresse(s) postale(s), etc.

SMS. Il est possible d'accéder aux SMS stockés sur le terminal en utilisant la Messaging API (MAPI) [15]. L'interface choisie correspond à celle des services réalisés lors du dump. La Messaging API permet

13. http://en.wikipedia.org/wiki/Personal_information_manager
14. http://msdn.microsoft.com/en-us/library/aa914277.aspx
15. http://msdn.microsoft.com/en-us/library/bb446118.aspx

de gérer les SMS et les e-mails. Le fonctionnement de cette API utilise un système de base de données propriétaire à Windows CE. Nous récupérons des pointeurs vers certaines tables, configurerons les champs souhaités puis nous récupérons effectivement ces champs. Nous avons implémenté cela pour les messages SMS présents dans la boîte de réception du smartphone. Pour chaque SMS, nous avons la date de délivrance, l'expéditeur et le contenu du SMS.

E-mails. Comme expliqué précédemment, il est également possible d'utiliser la Messaging API (MAPI) pour accéder aux e-mails stockés sur le terminal. Nous avons choisi d'implémenter la récupération des e-mails du smartphone qui ont été synchronisés avec l'ordinateur. Ils se trouvent dans le magasin "ActiveSync". Notons qu'il semble que cela s'applique également aux e-mails reçus par ActiveSync depuis un serveur Exchange. Cependant, cela n'a pas été testé par l'auteur. Le lecteur pourra se référer aux tutoriaux [25], [26] et [27] que nous avons utilisés pour notre implémentation pour plus de détails sur l'utilisation de MAPI pour les e-mails. Ceci peut paraître relativement complexe pour quelqu'un qui ne connaît pas le fonctionnement des APIs de la Messaging API. Cependant, elle s'avère relativement puissante avec l'utilisation et permet d'obtenir tous les champs souhaités. Nous avons une granularité sur les éléments récupérés. Notons qu'il serait également possible de récupérer les pièces jointes. Nous avons ainsi l'expéditeur, les destinataires multiples, la date d'envoi, le sujet et le contenu de l'e-mail.

Journaux des appels. La fréquence des appels téléphoniques peut également intéresser l'attaquant. La Phone API [16] permet de récupérer ces informations : date d'appel, durée, numéro de téléphone (ou nom) de l'appareil distant, sens de l'appel (reçu ou émis).

Informations de géolocalisation

Coordonnées GPS. Il est possible d'accéder aux informations GPS en utilisant le GPS Intermediate Driver [17]. Ceci permet de s'abstraire du matériel sous-jacent.

16. http://msdn.microsoft.com/en-us/library/bb416477.aspx
17. http://msdn.microsoft.com/en-us/library/bb202086.aspx

L'initialisation du GPS peut nécessiter plusieurs minutes. De plus, le smartphone peut se situer dans certains environnements où il lui est impossible de récupérer les informations. Typiquement, il peut être impossible de récupérer les coordonnées GPS si le smartphone se trouve à l'intérieur d'un bâtiment. C'est pourquoi nous utiliserons l'interface gérant les services en background. Le lecteur pourra également se référer au très bon article de Krishnaraj Varma [28] que nous avons utilisé.

La méthode implémentée n'effectue en aucun cas du polling. Une implémentation consistant à régulièrement faire appel à la fonction `GPSGetPosition` jusqu'à ce que l'on ait une réponse intéressante viderait la batterie beaucoup plus rapidement.

Il est important de noter ici que l'utilisation du GPS semble invisible aux yeux de l'utilisateur. Ainsi, notre rootkit peut utiliser ces APIs sans que l'utilisateur ne s'en aperçoive. Il pourrait être intéressant de notifier l'utilisateur lorsque le GPS est activé. De plus, une solution consistant à désactiver physiquement le GPS (par un bouton hardware) semble nécessaire pour protéger la vie privée des utilisateurs. Ces deux mécanismes n'existaient pas sur le terminal utilisé (HTC Kaiser).

Cellule GSM. Les tests que nous avons réalisés sur le HTC Kaiser concernant la récupération des coordonnées GPS présentent deux inconvénients. D'une part, l'utilisation du GPS a tendance à vider la batterie beaucoup plus vite qu'une utilisation normale, ce qui serait détecté par n'importe quel utilisateur. D'autre part, cette récupération a tendance à être longue, pouvant durer plusieurs minutes, même dans un environnement extérieur.

Pour ces raisons, les informations de cellule GSM présentent une très bonne alternative. Le réseau GSM est un réseau cellulaire, i.e. formé de cellules, chaque cellule étant une zone couverte par une antenne appartenant au réseau de l'opérateur. Lorsqu'un téléphone est connecté au réseau, il est typiquement en communication avec une antenne particulière. Lorsque l'utilisateur du téléphone se déplace, le téléphone se connecte à l'antenne la plus proche.

Le smartphone a donc bien évidemment connaissance à n'importe quel instant de l'antenne à laquelle il est relié. Il connaît notamment les informations suivantes :

- MCC (Mobile Country Code) : code correspondant au pays ;
- MNC (Mobile Network Code) : code correspondant à l'opérateur (ex. : Orange, Bouygues Telecom, SFR pour la France) ;
- LAC (Location Area Code) : code identifiant une certaine zone, propre à l'opérateur ;
- CID (Cell ID) : code identifiant l'antenne.

Les opérateurs ne publient pas de listing sur les localisations de leurs antennes. Cependant, des bases de données peuvent être constituées afin de donner une correspondance entre ces antennes et des zones approximatives (de quelques kilomètres en zone rurale à quelques centaines de mètres en zone urbaine). Ceci représente donc une méthode alternative aux coordonnées GPS et apparaît bien plus réaliste pour une attaque furtive.

Code PIN La méthode consistant à installer un hook au niveau des communications entre le CPU et le baseband permet également de récupérer le code PIN et ce, indépendamment de la sécurité implémentée par l'interface graphique permettant de le saisir. En effet, celui-ci passe en clair comme le montre le listing 1.4.

```
AT+CPIN="1234"
```

Listing 1.4. Commande AT d'envoi du code PIN

Ceci soulève le problème du niveau de confiance que l'on donne à ce code PIN. De nombreuses applications sont actuellement pensées pour utiliser la carte SIM (et son code PIN associé) : paiement en magasin, en ligne, etc.

Autres informations Des informations spécifiques au smartphone peuvent être expatriées : marque et révision du téléphone (ex : HTC, 22.45.88.07), version du système d'exploitation (ex : Windows CE 5.2.1620), identifiant unique au smartphone (IMEI), identifiant unique de la carte SIM (IMSI), etc.

Il est également possible d'effectuer une copie d'écran du smartphone pour espionner ce que fait l'utilisateur.

4 Conclusion

De plus en plus d'antivirus sont développés pour la plateforme Windows Mobile. Malheureusement, compte tenu du fait qu'il y a relativement peu de virus pour cette plateforme, leur base de signature est très limitée. Par exemple, à l'heure actuelle, Airscanner Mobile Antivirus ne possède que 10 virus dans sa base. Le rootkit développé n'a pas été détecté par les antivirus testés : Airscanner Antivirus, BitDefender Mobile Security, BullGuard Mobile Anti-virus. Windows Mobile 6, basé sur Windows CE 5, implémente des mécanismes de sécurité, mais reste très permissif. Ainsi, une application malveillante s'exécutant pourra accéder à toutes les informations présentes sur le terminal. Elle pourra installer son propre certificat, configurer le terminal pour intercepter certains SMS, s'installer pour être démarrée automatiquement au prochain redémarrage. Il est regrettable que n'importe quelle application puisse accéder à la majorité des services sous Windows Mobile 6. En particulier, les coordonnées GPS sont accessibles et ne nécessitent aucune action de l'utilisateur.

Le rootkit implémenté utilise des techniques "classiques" adaptées au monde de l'embarqué. Le mécanisme de signature de binaires peut être outrepassé dès que l'exécution de code est possible. Les processus, clefs de registre et fichiers utilisés peuvent être masqués en modifiant les structures internes au noyau. Les moyens de communication utilisables sont plus variés que dans le monde des PC : 3G, Wi-Fi, ActiveSync, SMS. Malgré une furtivité accrue, le rootkit reste détectable si l'on sait où regarder. La batterie reste l'élément le plus remarquable concernant les smarphones.

La diversité des services offerts par les smartphones actuels et le nombre d'APIs Windows Mobile 6 permet de récupérer une quantité d'information considérable. Tout repose sur la connaissance de ces APIs. Les attaquants qui connaissent l'environnement Windows des PC n'auront sans doute pas de difficultés pour utiliser de telles APIs sous Windows Mobile 6.

Références

1. MSDN. Windows Mobile 6 Documentation. http://msdn.microsoft.com/en-us/library/bb158486.aspx, 10 2008.

2. Holly Stevens and Christy Pettey. Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008. http://www.gartner.com/it/page.jsp?id=910112, 03 2009.
3. ARM Official Web Site. http://www.arm.com/.
4. Éric Lacombe and François Gaspard. Les rootkits sous Linux : passé, présent et futur. *MISC (Multi-System & Internet Security Cookbook)*, 34, 11 2007. Available at http://www.ddj.com/architect/184405459.
5. Petr Matousek. Subverting Windows CE Kernel. In *Microsoft BlueHat 2007 Fall Session*, 09 2007. http://www.fnop.org/public/download/COSEINC/subverting_wince.ppt.
6. F-Secure. WinCE/FlexiSPY.A riskware description. http://www.f-secure.com/sw-desc/riskware_wince_flexispy_a.shtml, 2008.
7. ITP.net. Etisalat's BlackBerry patch designed for surveillance. http://www.itp.net/news/561962-etisalats-blackberry-patch-designed-for-surveillance, 07 2009.
8. Olivier Bloch. Windows CE 6.0 et Windows Mobile 6, deux systèmes différents ! http://blogs.msdn.com/obloch/commentrss.aspx?PostID=1747978, 02 2007.
9. Christian Forsberg (MSDN). Effective Memory, Storage, and Power Management in Windows Mobile 5.0. http://msdn.microsoft.com/en-us/library/aa454885.aspx, 03 2006.
10. Dmitri Leman. Spy : A Windows CE API Interceptor. *Dr. Dobb's Journal*, 10 2003. Available at http://www.ddj.com/architect/184405459.
11. Reed and Steve Stuff blog. Slaying the Virtual Memory Monster - part I. http://blogs.msdn.com/hegenderfer/archive/2007/08/31/slaying-the-virtual-memory-monster.aspx, 08 2007.
12. Reed and Steve Stuff blog. Slaying the Virtual Memory Monster - part II. http://blogs.msdn.com/hegenderfer/archive/2007/10/01/slaying-the-virtual-memory-monster-part-ii.aspx, 10 2007.
13. Douglas Boling. Windows CE .NET Advanced Memory Management. http://msdn.microsoft.com/en-us/library/ms836325.aspx, 08 2002.
14. Reed and Steve Stuff blog. More Memory for ALL in WM 6.x. http://blogs.msdn.com/hegenderfer/archive/2009/06/10/more-memory-for-all-in-wm-6-x.aspx, 06 2009.
15. FlexiSPY website. http://www.flexispy.com/.
16. RIM. RIM Customer Statement Regarding Etisalat / SS8 Software. http://www.blackberrycool.com/wp-content/uploads/2009/07/blackberry-customer-statement-july-17-2009.pdf, 07 2009.
17. Sheran A. Gunasekera. Analyzing the SS8 Interceptor Application for the BlackBerry Handheld. http://chirashi.zensay.com/wp-content/uploads/2009/07/Analyzing_the_SS8_Interceptor_Application_for_the_BlackBerry_Handheld.pdf, 07 2009.
18. Code Project. Automatically Starting Your Application on Windows Mobile. http://www.codeproject.com/KB/mobile/WiMoAutostart.aspx, 07 2008.
19. Christopher Fairbairn. Windows Mobile Tip : Creating shortcuts. http://www.christec.co.nz/blog/archives/213, 03 2008.

20. MSDN. Configuring the Process Boot Phase. http://msdn.microsoft.com/en-us/library/ms901773.aspx.
21. MSDN. Security Policy Settings for Windows Mobile 6. http://msdn.microsoft.com/en-us/library/bb416355.aspx, 08 2008.
22. Michael Becher, Felix Freiling, and Boris Leidner. On the Effort to Create Smartphone Worms in Windows Mobile. In *Proceedings of the 8th Annual IEEE Information Assurance Workshop*, 06 2007. http://pi1.informatik.uni-mannheim.de/filepool/publications/on-the-effort-to-create-smartphone-worms-in-windows-mobile.pdf.
23. Willem Jan Hengeveld. Itsme's webpage. http://www.xs4all.nl/~itsme/, 2009.
24. Charlie Miller and Collin Mulliner. SMS Security Research Page. http://mulliner.org/security/sms/, 2009.
25. Windows Mobile Team Blog. Getting Started with MAPI. http://blogs.msdn.com/windowsmobile/archive/2007/03/21/getting-started-with-mapi.aspx, 03 2007.
26. Windows Mobile Team Blog. Practical Use of MAPI. http://blogs.msdn.com/windowsmobile/archive/2007/04/23/practical-use-of-mapi.aspx, 04 2007.
27. Raffaele Limosani. Programmatically retrieve mail BODY. http://blogs.msdn.com/windowsmobile/archive/2007/03/21/getting-started-with-mapi.aspx, 09 2008.
28. Krishnaraj varma's blog. Programming GPS on Windows Mobile 5 or above. http://www.krvarma.com/?p=51, 09 2008.