UNIVERSITE LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Sécurité dans les smartphones

Mémoire présenté en vue de l'obtention
du grade de Licencié en Informatique

Nicolas SIMON

Année académique 2006-2007

Remerciements

Je tiens à exprimer ma gratitude à Monsieur Roggeman, Directeur de ce mémoire, pour ses conseils judicieux, pour le temps qu'il a consacré à la relecture de ce mémoire ainsi que son encadrement tout au long de l'évolution de ce travail.
Je voudrais également remercier mes parents pour les encouragements qu'ils m'ont prodigués tout au long de ce travail.

Table des matières

Chapitre 1

Introduction

Les *smartphones* combinent les fonctionnalités d'un téléphone mobile et d'un assistant digital personnel (PDA). Ces appareils sont devenus de plus en plus présents durant les dernières années, intégrant petit à petit différentes technologies de connexion réseau, comme le Wi-Fi, le Bluetooth, les réseaux cellulaires (GSM).

Ces appareils furent commercialisés pour la première fois fin des années nonante et leurs rapides évolutions en font presque aujourd'hui des vrais ordinateurs mobiles, intégrant un système d'exploitation, des programmes bureautiques et ludiques installables ainsi que différents moyens pour se connecter à des réseaux locaux ou cellulaires.

Il n'existe pas de traduction en français du mot *smartphone*, j'utiliserai donc le mot anglais dans ce mémoire. Il est d'ailleurs à noter que l'utilisation du mot smartphone dans ce mémoire est parfois générique. En effet, il est très difficile aujourd'hui de distinguer un smartphone d'un téléphone cellulaire évolué ou d'un assistant personnel possédant une fonction téléphonique.

Ces nouveaux appareils supportent des services et des fonctionnalités additionnelles, et les fournisseurs de contenus ont très vite profité de cet engouement pour fournir toutes sortes de services payants à la demande.

Malheureusement, le développement de ces appareils et de ces services a avant tout été dirigé par le marketing, se concentrant d'avantage sur de nouvelles fonctionnalités et négligeant la sécurité. Par conséquent, les smartphones font maintenant face à de nouveaux problèmes de sécurité qui sont soit hérités du monde des ordinateurs personnels, soit complètement nouveaux. Ces problèmes tirent directement leur source de l'intégration massive de fonctionnalités et de multiples technologies de connexion dans un seul appareil mobile, ainsi que la possibilité de modifications ou d'installation de

nouvelles applications. D'autres problèmes peuvent être causés par des services spécifiques aux smartphones, qui requièrent souvent une infrastructure et des logiciels complexes.

Le but de ce mémoire est de décrire ces nouveaux appareils et d'analyser les problèmes de sécurité auxquels ils doivent faire face. Ce mémoire est donc avant tout une collecte, une analyse et une mise en structure d'informations sur le sujet. Il n'a pas toujours été aisé de trouver toute l'information voulue. La documentation relative aux fonctionnalités et aux technologies qui composent les smartphones est actuellement fort limitée. De plus, un tel rassemblement d'informations sur ce sujet est quasi inexistant, et c'est en cela que mon travaille est original.

La rédaction de ce mémoire m'a apporté beaucoup de choses. Tout d'abord, la sécurité informatique est un domaine qui m'intéresse énormément, mais dont je ne connaissais pas grand chose. La rédaction de ce document m'a fait prendre conscience de l'importance sans cesse croissante de la sécurité des outils informatiques. De plus, la recherche de documentations, ainsi que l'analyse et la compréhension de celle-ci m'a apporté de nouvelles connaissances. Enfin, l'écriture d'un tel document scientifique, avec toute la critique et la rigueur qui en découle, fut une expérience totalement nouvelle.

La structure du mémoire est la suivante : le chapitre deux présente ces appareils mobiles, ce qu'ils sont, leurs technologies de connexions ainsi que les systèmes d'exploitation existants.

Le chapitre trois introduit le sujet principal de ce mémoire, à savoir l'étude de la sécurité de ces nouveaux appareils. J'y décris les différents aspects qui, du point de vue sécuritaire, les distinguent de la sécurité informatique conventionnelle. Il s'en suit un aperçu des différentes menaces possibles envers ces appareils.

J'ai choisi ensuite de me focaliser sur l'étude de trois domaines bien précis. Le chapitre quatre propose donc une description de la machine virtuelle Java spécifique à ces appareils mobiles, Java Micro Edition. Cet environnement de développement constitue un bon exemple pour illustrer les points communs ou les différences qu'il peut y avoir avec la machine virtuelle standard, présente sur la plupart des ordinateurs personnels.

Le chapitre cinq illustre le déploiement d'un *shellcode* sur un appareil mobile possédant Windows CE et utilisant une architecture ARM. Nous verront ici comment parvenir à un dépassement de tampon permettant d'exécuter du code. Ce type de vulnérabilité étant courant dans le monde informatique conventionnel, nous verrons que les smartphones n'y échappent pas, mais nous verrons aussi quelles sont les différences et les propriétés dont il

convient de tenir compte pour mener à bien ce type d'attaque sur un appareil mobile.

Le chapitre six présente un tout nouveau genre d'attaque possible envers les appareils mobiles, à savoir une attaque par services croisés, qui est propre aux appareils mobiles et tire directement parti de l'intégration de différentes interfaces de communications (dont certaines sont payantes). Ce type d'attaque est encore très peu documenté, et dans la deuxième partie de ce chapitre, je présenterai un dispositif permettant d'empêcher ce type d'attaque.

Le chapitre sept présente différentes vulnérabilités déjà investiguées par le passé dans des articles et dont je n'ai pas ou peu parlé dans ce mémoire ; les raisons sont multiples, soit je n'ai pas jugé intéressant d'en parler, soit je n'avais pas la place ou pas la documentation suffisante pour en parler.

Enfin le chapitre huit constitue la conclusion de ce mémoire. Nous verrons que tout est encore à faire dans le domaine de la sécurité informatique des appareils mobiles.

Chapitre 2

Appareils mobiles

Ce chapitre introduit la notion d'appareils mobiles et liste les différentes technologies sans fil.

2.1 Présentation

Les appareils mobiles, ou « mobiles », sont de petits engins électroniques portables. Les premiers appareils téléphoniques portables intégrant un organisateur furent appelés appareils mobiles, et apparurent pour la première fois vers la fin des années septante. Les appareils mobiles de l'époque n'avaient bien entendu rien en commun avec les appareils actuels, mis à part le fait de pouvoir effectuer des appels téléphoniques. De même, les premiers organisateurs n'avaient rien des Personal Digital Assistant (PDA) actuels, outre le fait de pouvoir enregistrer une adresse ou un rendez-vous. On peut dire qu'en général les premiers appareils mobiles furent conçus pour une tâche ou pour une application spécifique, grâce à l'association d'un logiciel propriétaire et d'un matériel dédicacé, alors que les appareils mobiles actuels sont conçus pour être polyvalents.

Pour atteindre cette flexibilité, beaucoup d'appareils mobiles accueillent un système d'exploitation permettant l'installation d'applications additionnelles. Les autres points clés sont l'intégration d'une connectivité réseau, l'augmentation de la puissance de calcul et des possibilités de stockage. De plus, beaucoup de PDA sont équipés de slots d'extension permettant l'ajout de matériel accessoire supplémentaire.

2.1.1 Types d'appareils mobiles

Jusqu'il y a plus ou moins cinq ans, les téléphones mobiles et PDA constituaient les seuls engins mobiles avec les lecteurs audio portables. Aujourd'hui beaucoup de mobiles différents existent. De nouveaux types d'appareils furent conçus pour répondre à certaines exigences des utilisateurs. Par exemple, les PDA doivent être compacts et posséder une longue autonomie, alors que les lecteurs multimédia mobiles ont besoin d'un grand espace de stockage, d'un processeur rapide et, dans le cas d'un lecteur vidéo, d'un grand écran. Combiner ces différents dispositifs dans un seul appareil n'est pas toujours possible, c'est pourquoi plusieurs types d'appareils mobiles distincts existent aujourd'hui.

Les appareils mobiles peuvent être classés en 8 catégories :
- Téléphone mobile
- PDA
- Smartphone
- Tablette PC
- Notebook
- Lecteur multimédia mobile
- Console de jeu mobile
- Appareil mobile industriel

Téléphone mobile

Les téléphones mobiles présentent aujourd'hui une typologie très variée et fournissent différents niveaux de fonctionnalités. Un simple téléphone mobile offre les fonctionnalités de base, tel que la possibilité d'effectuer un appel téléphonique et l'envoi d'un court message de texte. Cependant de nos jours, même le téléphone mobile le plus basique propose des fonctions supplémentaires comme une alarme ou un calendrier. Des téléphones mobiles plus évolués peuvent offrir des fonctions additionnelles comme la synchronisation du contenu du calendrier ou du répertoire téléphonique avec un ordinateur de bureau. La plupart des téléphones mobiles font tourner un système d'exploitation spécialisé et compact.

PDA

Les PDA sont des appareils de la taille d'un téléphone mobile, mais possédant généralement un large écran tactile à la place d'un écran plus petit muni d'un clavier. Les PDA actuels disposent de processeurs relativement rapides, mais d'une capacité mémoire et de stockage plutôt limitée. Le cœur de chaque PDA est un ensemble de logiciels pour la gestion des informations personnelles (personal information management, PIM). Cet ensemble est au moins composé d'un carnet d'adresses, d'un calendrier et d'un petit traitement de texte. Une des caractéristiques les plus importantes des PDA actuels est qu'ils font tourner un système d'exploitation supportant l'installation de logiciels supplémentaires. Beaucoup de PDA récents sont aussi équipés d'un affichage haute résolution et fournissent généralement une certaine connectivité sans fil.

Smartphones

Un smartphone est une combinaison d'un téléphone mobile et d'un PDA. Fondamentalement il existe deux variantes différentes : une version élaborée ressemblant à un PDA et une version plus simple ressemblant à un téléphone mobile. Un smartphone possède beaucoup d'applications communes à un PDA, il supporte aussi l'installation d'applications supplémentaires. Les dernières générations de smartphones offrent généralement des possibilités de gestion de réseau sans fil.

Tablette PC

Les tablettes PC sont pour la plupart des écrans tactiles mobiles sans clavier, supportant une connectivité sans fil pour la visualisation de contenu multimédia et de documents en ligne. La majorité des tablettes sont construites en utilisant des composants standards d'ordinateurs personnels, et donc font tourner un système d'exploitation commun aux ordinateurs personnels qui peut être enrichi de quelques services d'interface spécifiques.

Notebook

Les notebooks sont de petits ordinateurs portables. Souvent ils ne disposent pas d'un clavier complet, mais ils peuvent avoir des fonctionnalités supplémentaires, comme un écran tactile ou une souris « Touchpad ». Beaucoup de notebooks font tourner les systèmes d'exploitation standards des

ordinateurs personnels, tandis que d'autres utilisent un système d'exploitation spécialisé, plus léger. Ce type d'appareil se situe normalement entre un ordinateur portable et un PDA, en terme de taille et de fonctionnalités. Néanmoins à l'heure actuelle, on ne distingue plus bien un notebook d'un ordinateur portable, et le terme *notebook* est maintenant communément employé pour désigné un ordinateur portable.

Lecteur multimédia mobile

Les lecteurs multimédia sont des appareils spécialement conçus pour accéder à du contenu multimédia. Dans les versions les plus basiques, ils sont aussi appelés *lecteurs de musique* ou *baladeurs*. Les modèles haut de gamme peuvent inclure un lecteur et enregistreur vidéo portable. Les modèles récents incorporent une connectivité sans fil pour accéder à du contenu à travers un réseau. La plupart des lecteurs utilisent un système d'exploitation personnalisé et ne supporte pas l'installation de logiciels supplémentaires.

Console de jeu mobile

Les consoles de jeu mobiles sont conçues pour jouer à des jeux vidéos. La plupart de ces appareils sont aussi capables de fournir du contenu multimédia. Ceci entrave donc parfois la distinction entre une console de jeu portable et un lecteur multimédia mobile. Les consoles récentes utilisent une connectivité sans fil pour supporter les jeux multi-joueurs. Comme les lecteurs multimédias, la plupart des consoles de jeux portables font tourner un système d'exploitation personnalisé et ne supportent pas l'installation de logiciels supplémentaires autre que des jeux, bien qu'il existe des outils permettant de les utiliser comme un ordinateur portable.

Appareil mobile industriel

Les appareils mobiles *industriels* sont des engins destinés à une application commerciale, médicale ou militaire spécifique. Ces appareils ne possèdent pas une grande base d'utilisateurs et sont habituellement équipés de logiciels simples et personnalisés.

2.1.2 Description matérielle

Le matériel utilisé par la plupart des appareils mobiles est fondamentalement différent du matériel utilisé par les ordinateurs personnels. Les raisons sont les besoins spécifiques en taille et fonctionnalités, notamment au point de vue de l'efficacité énergétique.

La plupart des appareils mobiles sont basés sur des processeurs dits *applicatifs*, qui incluent non seulement une unité centrale (central processor unit, CPU), mais aussi les contrôleurs mémoires et périphériques. Les processeurs applicatifs sont des plateformes complètes qui sont adaptées par le fabricant du mobile. Ces adaptations incluent la taille de la mémoire, l'affichage, la connectivité (LAN sans fil par exemple), ainsi que des processeurs de signaux spécialisés (digital signal processor, DSP) pour le traitement multimédia.

Les processeurs applicatifs les plus répandus utilisent des microprocesseurs basés sur l'architecture ARM (Advanced RISC Machine). ARM est une architecture RISC (Reduced instruction set computer) 32 bits très performante et peu consommatrice, conçue spécialement pour des appareils intégrés.

Deux processeurs applicatifs largement utilisés dans les appareils mobiles sont l'Intel XScale et le TI-OMAP de Texas Instrument. Ces 2 processeurs sont basés sur des microprocesseurs ARM et incluent des interfaces matérielles comme : USB (Universal Serial Bus), PCMCIA (Personal Computer Memory Card International Association), cartes mémoires, Bluetooth, LAN sans fil (Wi-Fi) et réseaux cellulaires.

La partie matérielle gérant le fonctionnement cellulaire du mobile est intéressante, car c'est un *système dans le système* qui contient son propre CPU séparé et fait tourner son propre logiciel. Cette séparation du système principal est faite pour économiser l'énergie et réduire la charge du système principal. En effet, la partie cellulaire doit être constamment alimentée afin de garder la connexion avec l'infrastructure, alors que le système principal doit seulement être lancé durant les interactions avec l'utilisateur. En fait les deux systèmes sont rattachés de manière très étroite, et chacun possède un contrôle partiel sur l'autre (le système principal peut éteindre la partie cellulaire, et la partie cellulaire peut réveiller le système principal).

2.2 Technologies sans fil

Les technologies sans fil jouent un rôle important dans le domaine des appareils mobiles. Cette section montrera d'abord pourquoi et comment les technologies sans fil jouent un rôle si important dans le monde des appareils mobiles, et ensuite il présentera les différentes technologies sans fil.

2.2.1 Le rôle du sans fil

Les technologies sans fil n'ont pas seulement tenu un rôle important dans l'usage des appareils mobiles, ils ont également changé entièrement le concept d'appareils mobiles dans une direction qu'aucune technologie n'avait suivie auparavant.

Les téléphones mobiles n'existeraient pas sans ces technologies, mais les technologies de communication de données évoluées comme le GPRS (General Packet Radio Service) ont eu un effet majeur sur les fonctionnalités des téléphones mobiles, car ils constituent la base des services comme le MMS (Multimedia Messaging Service) et l'accès Internet sur le mobile. Quand on observe un PDA ou un appareil de téléphone mobile, l'impact de la connectivité sans fil est encore plus évidente. Un PDA équipé d'une technologie sans fil peut transformer un simple calendrier et carnet d'adresses en un petit bureau mobile, utilisable partout et tout le temps. D'autres exemples peuvent être la connexion d'un mobile à un réseau sans le besoin d'utiliser un cable, le suivi automatique d'email vers un mobile ...

Les appareils avec de multiples interfaces sans fil, comme les smartphones, montrent particulièrement bien le rôle majeur des technologies sans fil. Pour un smartphone capable de se connecter à un réseau LAN sans fil et à un réseau téléphonique mobile, il devient très aisé d'être connecté en permanence, et donc de pouvoir avoir accès à des ressources en ligne de manière très facile.

De nos jours, un appareil mobile sans aucune connectivité sans fil sera considéré comme pratiquement inutile (une exception notable sont les baladeurs multimédia). La technologie sans fil est omniprésente et les consommateurs veulent accéder à Internet en tout lieu et tout temps. Cela a pour conséquence d'accroître la demande pour des appareils possédant une interface sans fil. À côté des utilisateurs, les compagnies ont largement adopté les appareils sans fil permettant à leurs employés de rester en contact avec leur bureau ou pour aider les techniciens à garder un oeil sur les infrastructures IT de la société en dehors des heures de bureau et pendant le week end.

Les nouveaux services comme le suivi d'email ou la vidéo conférence a même amené ces appareils au niveau des dirigeants de grandes compagnies et spécialement auprès de personnes qui autrement n'envisageraient que d'utiliser un simple téléphone mobile.

En résumé, les technologies sans fil ont un impact majeur dans le monde des appareils mobiles. Ils augmentent l'attractivité des appareils mobiles pour un plus large groupe de personnes et ils augmentent la fréquence d'utilisation de ces appareils.

2.2.2 Technologies de télécommunication sans fil

Les technologies de télécommunication sans fil peuvent être classées en deux catégories : les technologies pour la téléphonie mobile, comme le GSM et le CDMA, et les technologies utilisées pour la communication de données, comme le GPRS et EVDO. Ces deux catégories utilisent la même infrastructure réseau.

La télécommunication mobile sans fil a évolué au travers du temps pour fournir des services additionnels et une plus large bande passante pour la communication de données. De plus, les technologies de seconde génération actuelles (2G) sont petit à petit remplacées par des technologies de troisième génération (3G).

Il existe de très nombreuses technologies de télécommunication sans fil, suivant la zone géographique ou la génération dont on parle, et les énumérer toutes ne serait pas possible dans ce mémoire. Ne seront présentées ici que les technologies majeures présentes en Europe et aux Etats-Unis, et bien souvent aussi dans la grande majorité de l'Asie. Celles-ci sont utilisées activement de nos jours dans les infrastructures et les appareils mobiles.

GSM

Le Global System For Mobile Telecommunication (GSM) est le standard le plus populaire pour les systèmes de télécommunication mobile dans le monde. Ce service, conçu en Europe par l'ETSI (European Telecommunications Standards Institute) est utilisé par 2 milliards de personnes dans 212 pays différents, majoritairement en Europe, Asie et partiellement aux Etats-Unis. Outre l'offre basique de communication voix et donnée, le GSM offre des services comme le Short Message Service (SMS) pour envoyer et recevoir de courts messages de texte sur son téléphone mobile.

GPRS

Le General Packet Radio Service (GPRS) est une technologie de communication de données pour le GSM. Elle constitue une technologie souvent décrite comme *always-on* car le prix de l'usage du service GPRS est calculé suivant le volume de trafic transféré au lieu du temps de connexion. Les utilisateurs peuvent donc rester en ligne tout le temps et doivent seulement acquitter le volume de données transférées. Typiquement la vitesse de connexion du GPRS se situe entre 20 et 80 kbit/s (le maximum théorique étant de 160 kbit/s), alors que le GSM fournit seulement un débit de 14.4 kbit/s. Le GPRS constitue aussi une base pour de très nombreux services additionnels offerts par les opérateurs mobiles modernes, comme le MMS.

EDGE

L'Enhanced Data rates for GSM Evolution (EDGE) est une technologie de transmission de données pour les réseaux mobiles. C'est une évolution du GPRS (EDGE reste compatible avec lui), dont le but est principalement d'augmenter la vitesse de transmission. EDGE supporte théoriquement des débits jusqu'à 473,6 kbit/s. EDGE peut être classé comme 2,5G ou 3G en fonction du débit.

CDMA

Le Code Division Multiple Access est une autre technologie de télécommunication mobile et est largement répandue dans le monde, sauf en Europe. CDMA est une technologie similaire au GSM et offre plus ou moins les mêmes fonctionnalités. La vitesse maximum de communication des données est de 9,6 kbit/s.

EVDO

L'Evolution-Data Optimized (EVDO, aussi appelé CDMA2000) est une technologie avancée de transmission de données pour les réseaux CDMA. Comme le GPRS, il s'agit d'une technologie de connexion permanente au réseau de paquet, et dont la facturation est basée sur le volume de trafic transféré. EVDO est une technologie de troisième génération (3G) et offre des débits allant jusqu'à 3,1 Mbit/s (une révision récente de la technologie - EVDO Rev B - permet d'atteindre un débit théorique de 4,9 Mbit/s en transmission descendante).

UMTS

L'Universal Mobile Telecommunication System (UMTS) est une technologie de télécommunication mobile de troisième génération (concurrente de CDMA2000), principalement présente en Europe et en Asie. Avec la technologie HSDPA (High-Speed Downlink Packet Access), le débit maximum descendant peut atteindre 14,4 Mbit/s, bien que les débits actuels courants soient de l'ordre de 384 kbit/s à 3,6 Mbit/s suivants les technologies supportées par les appareils mobiles utilisant l'UMTS.

Réseau sans fil IEEE 802.16

Le 802.16 ou sa dénomination commerciale, le WiMaX (Worldwide Interoperability for Microwave Access) est un standard de réseau sans fil métropolitain. L'objectif du WiMAX est de fournir une connexion internet à haut débit sur une zone de couverture de plusieurs kilomètres de rayon. Le WiMaX utilise des canaux sur une très large bande de fréquences, de 2 à 66 GHz, dans laquelle on trouve des technologies existantes, comme le Wi-Fi, et qui autorise des débits, des portées et des usages très variés. Cette multiplicité de bandes de fréquences, des débits, des portées et usages possibles est d'ailleurs un atout pour le WiMaX : selon l'angle choisi, le WiMAX est tour à tour un simple prolongement du Wi-Fi (le Wi-Fi du futur), le cœur de réseau du Wi-Fi, ou mieux, la convergence du Wi-Fi et du réseau cellulaire de troisième génération (comme l'UMTS). Enfin le WiMaX est vu comme une technologie de quatrième génération.

2.2.3 Réseaux locaux personnels et sans fil.

Durant les dernières années, les réseaux locaux sans fil (WLAN) sont devenus très répandus dans notre vie quotidienne (à la maison, au bureau ...)

Toutes les technologies sans fil ne furent pas développées seulement pour l'accès réseau, certaines sont juste destinées à remplacer les cables quand on désire connecter des appareils. Ces technologies ont évolué dans le temps. Ci-dessous seront présentées les technologies les plus communes et utiles pour les appareils sans fil, c'est-à-dire : le DECT, le wireless LAN (IEEE 802.11), le Bluetooth, l'infrarouge et l'Ultra Wide Band.

DECT

Le DECT (Digital Enhanced Cordless Telephone), anciennement Digital European Cordless Telephone, est une norme de téléphonie sans-fil numérique destinée aux particuliers comme aux entreprises sur la gamme de fréquence 1880 à 1900 MHz en Europe. Cette norme, même si elle a été conçue pour une gamme large d'utilisations, est aujourd'hui principalement utilisée pour des communications vocales avec un téléphone domestique, à portée limitée, qui permet des déplacements dans une habitation ou une entreprise (portée maximum de 100 à 300 m, en terrain dégagé).

Réseau sans fil IEEE 802.11

Le IEEE 802.11 est le standard de facto pour les réseaux sans fil, et est souvent appelé Wi-Fi (wireless fidelity) ou WLAN (wireless local area network). Le standard 802.11 ne permettait d'atteindre qu'une vitesse de 2 Mbit/s, alors que son successeur, le 802.11b permet d'atteindre 11 Mbit/s. La version la plus récente, le 802.11g, permet d'atteindre 54 Mbit/s (une prochaine évolution, le 802.11n, permettra d'atteindre 100 Mbit/s). Toutes les implantations de ces technologies opèrent dans la bande de fréquence libre de 2,4 GHz. (une variante, le 802.11a, opère dans les 5 Ghz)

Le 802.11 a deux modes de fonctionnement, le mode avec station de base et le mode ad hoc. Dans le mode avec station de base, un élément du réseau, la station de base (ou point d'accès - access point), coordonne l'accès au réseau parmi les utilisateurs associés. Dans le mode ad hoc, tous les appareils connectés au réseau ont un contrôle équivalent sur le spectre, et peuvent donc interférer les uns avec les autres pour l'accès au spectre. Le mode station de base est le mode le plus utilisé.

Le spectre wireless est un média partagé : tous les nœuds dans le champ d'émission de l'émetteur peuvent écouter les transmissions. Il y a donc lieu d'introduire de nombreux mécanismes de sécurité basés sur le chiffrement. Les mécanismes de sécurité ont deux fonctions : la première est de restreindre l'utilisation du réseau qu'aux appareils (clients) autorisés ; la seconde est de fournir la confidentialité des données. Le Wired Equivalent Privacy (WEP) fut le premier mécanisme de sécurité sans fil implanté, mais il fut démontré comme étant insécurisé et facile à casser (voir [15]). Le WEP a été remplacé par des mécanismes plus robustes, comme le Wi-Fi Protected Access (WPA) et WPA2.

Bluetooth

Le Bluetooth est une technologie de communication sans fil de courte portée. Elle est principalement utilisée pour interconnecter de petits appareils comme les PDA, téléphones, imprimantes, clavier et souris. Une des motivations du Bluetooth est de fournir l'interopérabilité entre appareils de différents constructeurs.

Le Bluetooth fut spécifié par le Bluetooth SIG (Special Interest Group) et définit plusieurs *profils*. Un profil est un service spécifique qui définit toutes les composantes de ce service, du protocole radio bas niveau à l'implantation de l'application. Le Bluetooth, comme le 802.11, travaille dans la bande des 2,4 GHz, et donc ces deux technologies peuvent interférer l'une avec l'autre. La vitesse de transmission va de 1 mbit/s (pour la version 1.2) à 3 mbit/s (pour la version 2.0).

Un point important à propos du Bluetooth est qu'il ne constitue pas une technologie réseau par définition. Il peut être utilisé *pour faire du réseau*, mais à l'opposé du 802.11, son but est l'interconnexion d'appareils. Le Bluetooth spécifie de nombreux protocoles de transport et services de découverte, et est donc plus qu'une simple technologie d'infrastructure réseau. Il fut créé avec une attention particulière en matière de sécurité, le modèle Bluetooth est donc considéré a priori comme sécurisé.

L'infrarouge

L'infrarouge ou IrDA fut une des premières technologies permettant une communication sans fil pour les premiers PDA et téléphones mobile. Il supporte la communication bas débit (de 2,6 kbit/s à 16 Mbit/s), point à point, entre deux appareils dont le champ de vision coïncide. IrDA spécifie quelques formats d'échanges de données qui furent plus tard repris par le Bluetooth. En général, IrDA peut être considéré comme sécurisé car la communication entre deux appareils exige une distance très courte (typiquement un mètre) et un champ de vision direct entre les 2 émetteurs/récepteurs. Cependant, aucun mécanisme de sécurité n'est implanté.

Ultra Wide Band

L'UWB (Ultra Wide Band) peut être utilisé en tant que technique de communication sans fil, qui fournit des taux de transferts réseaux très élevés sur des distances relativement courtes et à faible puissance. Bien que la vitesse

de communication décroisse rapidement en fonction de la distance, l'UWB serait capable de remplacer les systèmes filaires actuels.

Ce niveau de performances, l'UWB le doit à son mode d'émission des signaux. Elle n'utilise pas de fréquence porteuse, mais envoie des impulsions électroniques de très courte durée (de l'ordre de la picoseconde), de faible puissance (de 50 à 70 mW contre 100 mW pour le Wi-Fi et le Bluetooth) et sur une gamme de fréquences très larges (de 3,1 GHz à 10,6 GHz). Ceci lui permet de ne pas interférer avec les normes 802.11 ou Bluetooth, qui toutes évoluent dans la bande des 2,4 GHz.

Autre atout, l'UWB offre théoriquement de meilleures garanties que le Wi-Fi en matière de sécurité. Enfin, l'UWB autorise l'établissement d'une connexion très rapide, de l'ordre de la seconde, entre le périphérique et le poste auquel il est rattaché.

2.3 Systèmes d'exploitation pour mobiles

Les systèmes d'exploitation pour mobiles sont différents de ceux développés pour les ordinateurs personnels. Les raisons de ces différences sont les contraintes d'énergie et d'espace mémoire associés à du matériel spécifique, et des contraintes comme le support d'un usage interactif et sporadique. Par exemple, le système d'exploitation d'un PDA peut nécessiter une fonction matérielle de réveil pour implanter des fonctions comme des rappels (pour se souvenir d'un rendez-vous par exemple), qui peut se produire quand l'appareil est en veille pour économiser son énergie.

D'autres différences sont induites par la partie logiciel du système plutôt que par le noyau du système d'exploitation. Les différences du système logiciel tirent leur origine des contraintes très différentes de l'interface utilisateur de ces appareils, comme l'absence d'un clavier complet et l'affichage relativement de petite taille. L'interface utilisateur sera discutée au point 2.4. Les systèmes d'exploitation les plus courants seront présentés ci-après.

2.3.1 Systèmes d'exploitation courants

Il y a beaucoup de systèmes d'exploitation pour mobiles. Seules les systèmes d'exploitation les plus courants utilisés par les PDA et smartphones seront présentés ici. Fondamentalement, seulement 4 systèmes d'exploitation principaux pour appareils mobiles existent : Symbian OS, PalmOS, Linux et Windows CE.

2.3.2 Symbian OS

Symbian OS est le système d'exploitation pour smartphones le plus populaire, utilisé par de nombreux fabricants de téléphones, comme Nokia, Panasonic, Samsung, Siemens ou SonyEricsson. SymbianOS est issu de la famille des systèmes d'exploitation EPOC qui fut développée par Psion dans les années 90 pour leurs PDA. Symbian OS existe depuis environ 2001 et tourne exclusivement sur les processeurs ARM. Il fut créé pour fournir un système d'exploitation standard pour smartphones qui peut être utilisé par différents fabricants. L'idée sous-jacente était de réduire le temps de développement et les coûts lors de la création d'un nouveau smartphone tout en étant toujours capable de personnaliser le système d'exploitation pour refléter les choix graphiques d'un fabricant d'appareils mobiles.

Symbian OS est basé sur un design de *micro kernel* (le noyau ne comporte que le minimum nécessaire) et implante la plupart des fonctions que l'on trouve actuellement sur un système d'exploitation comme : un système préemptif multi-tâches et *multi-threads*, un gestionnaire du système de fichier et une protection de la mémoire. Le système d'exploitation ne requiert donc qu'une petite puissance CPU et peu d'espace de stockage et est fondamentalement développé pour économiser l'énergie. Symbian OS est mono-utilisateur et distingue seulement le mode utilisateur du mode kernel.

Symbian OS fut la cible de plusieurs virus, le plus connu étant Cabir. Généralement ces virus s'envoient eux-mêmes d'un téléphone à un autre par Bluetooth. Jusqu'à présent, aucun virus n'a tiré parti d'une quelconque faille dans l'OS, ceux-ci se répandant grâce aux utilisateurs acceptant d'installer un logiciel malgré l'avertissement comme quoi celui-ci n'est pas digne de confiance. Cependant, la majorité des utilisateurs ne devrait pas devoir se soucier cette situation, c'est pourquoi la version 9 de Symbian OS a adopté un *capability model*. Ainsi les logiciels installés ne peuvent théoriquement pas causer de dommages (comme envoyer de manière cachée des données ou des messages facturés à l'utilisateur) sans être signés numériquement. Les développeurs peuvent demander d'avoir leur logiciel signé via le programme *Symbian Signed* (moyennant finance). Néanmoins ceux-ci ont toujours la possibilité de signer eux-même leurs programmes.

Historique

- Symbian OS v5 ou EPOC Release 5. Première version de l'OS nommée *Symbian*.
- Symbian OS v5.1 ou ER5u. Support de l'unicode.

- Symbian OS v6.0 et v6.1. Premier téléphone *ouvert* (dans le sens où l'on pouvait installer des logiciels), le Nokia 9210 sort avec la version 6.0.
- Symbian OS v7.0 and v7.0s (2003). Version majeure sur laquelle beaucoup d'interfaces utilisateurs ont été développées : UIQ (Sony Ericsson), Series 80 (Nokia), Series 90 (Nokia).
- Symbian OS v8.0. (2004). Cette version introduit un nouveau cœur (EKA2) radicalement différent dans son fonctionnement de l'EKA1, offrant pour la première fois un vrai support matériel en temps-réel.
- Symbian OS v8.1. Amélioration de la v8.0 disponible en 8.1a qui continue de supporter EKA1 et 8.1b qui supporte EKA2.
- Symbian OS v9.1 (2004). Version utilisée uniquement en interne chez Symbian.
- Symbian OS v9.1 (2005). Cette version met l'accent sur la sécurité. Seules les applications signées peuvent accéder aux fonctionnalités critiques du téléphone.
- Symbian OS v9.2 (premier trimestre 2006). Ajout du support du Bluetooth 2.0 et du protocol Open Mobile Alliance Device Management.
- Symbian OS v9.3 (mi 2006). Ajout du support natif du 802.11 et du HSDPA (High-Speed Downlink Packet Acces - protocol pour échange de paquets pour la téléphonie de troisième génération).
- Symbian OS v9.5 (mars 2007). Amélioration de l'usage de la mémoire vive (diminution de 25% de l'usage de la RAM), optimisation du temps de lancement des applications et support en natif de la télédiffusion numérique mobile (norme DVB-H et ISDB-T).

2.3.3 PalmOS

PalmOs fut développé par Palm Inc et plus tard par PalmSource comme le système d'exploitation pour les PDA de la série des PalmPilot. PalmOS existe depuis 1996 et fut à l'origine uniquement utilisé par Palm lui-même. Plus tard, Palm commença a accorder des licenses d'usage de son OS à d'autres fabricants d'appareils.

La première version de PalmOS fut développée pour tourner sur un matériel très limité, et donc ne fournissait pas de fonctions comme le support multi-tâches, hormis certaines tâches spécifiques du système d'exploitation. Une autre différence avec les systèmes d'exploitation traditionnels est le manque de protection mémoire. Depuis que PalmOs est aussi utilisé dans les smartphones, des fonctions supplémentaires furent introduites, comme un meilleur support multi-tâches. Avec la version 5, Palm passa des proces-

seurs Motorola 68k aux processeurs ARM pour ses appareils, mais garda une émulation du 68k pour assurer une compatibilité ascendante. De plus, avec PalmOS 5, Palm introduisit des mécanismes de sécurité, et fournit à présent le premier système d'exploitation pour appareil mobile avec un chiffrement du système de fichier en natif (built in).

Historique

- Palm OS v1.0 à 4.1 est basé sur un petit noyau sous license par Kodak. Alors que ces versions sont techniquement capables de multi-tâches, les termes de la licence interdisaient à Palm de publier l'API pour créer ou manipuler les tâches dans l'OS.
- Palm OS v5.0 (novembre 2002) introduit le support pour les processeurs ARM. Décrit comme une avancée majeure pour le support de ces processeurs, les applications existantes peuvent néanmoins toujours être lancées dans un environnement émulé. C'est aussi vers cette époque que Palm scinda sa partie matérielle de sa partie système d'exploitation, créant PalmSource Inc (pour l'OS) et palmOne (renommé plus tard Palm Inc).
- Palm Os v5.2 introduit Graffiti 2, une nouvelle version de cet outil puissant de reconnaissance de caractères écrits avec un stylet.
- Palm OS v5.4 introduit le système de fichier non volatile (NVFS - Non-Volatile File System), et utilise à présent la mémoire flash pour stocker les données, à la place de la DRAM (Dynamic random access memory), empêchant la perte des données en cas de panne de batteries.
- Palm OS v6 fut introduit sur le marché en Janvier 2004, tout en continuant le développement de Palm OS v5. Il fait tourner des applications compilées pour processeur ARM en natif et améliore grandement le support multimédia. En février 2004 Palm OS 5 devient Palm Garnet et Palm OS 6 devient Palm OS Cobalt.
- Palm OS v6.1 vit le jour en septembre 2004. Celui-ci étend son support à une large variété de résolutions LCD, redessine la navigation *à une main* de ces appareils et ajoute le support *multi-thread* pour les applications. Cependant cette version ne sortira jamais pour le grand public.
- En septembre 2005, PalmSource fut racheté par ACCESS, et le développement futur de son OS reposera sur un noyau Linux (du nom de *ACCESS Linux Platform*)

2.3.4 Linux

Linux possède une longue histoire dans les mondes des ordinateurs personnels et est utilisé dans toutes sortes d'appareils. Ces dernières années, Linux fit son entrée comme système d'exploitation dans le monde des Smartphones. C'est un système d'exploitation complet, incluant toutes les fonctionnalités nécessaires, comme le support multi-tâches, la protection mémoire et le support multi-utilisateurs.

En général, Linux a divers avantages sur ses concurrents, mais aussi des désavantages, car il fut à l'origine développé pour les ordinateurs personnels et non pour les appareils mobiles. Le principal désavantage est le manque de mécanisme avancé pour la sauvegarde de l'énergie, et le besoin assez conséquent de mémoire, que ce soit pour l'exécution ou le stockage. Linux est utilisé de façon très diverse comme système d'exploitation, et seuls le noyau et les bibliothèques restent inchangés d'un appareil à un autre. Linux ne fournit pas non plus d'interface graphique standard. A la place, chaque fabricant choisit un des nombreux systèmes disponibles ou développe sa propre plateforme.

MontaVista, Green Hills Software, Wind River Systems ou LynuxWorks sont des exemples de sociétés spécialisées dans la conception de systèmes d'exploitation basé sur le noyau de Linux et destinés à être utilisés dans des systèmes embarqués ou des appareils mobiles. Les fabricants de smartphones, par exemple, font appel à ces sociétés pour leur fournir un système d'exploitation pouvant fonctionner sur leurs matériels.

Si nous prenons le cas de MontaVista, cette société intègre dans son système d'exploitation destiné aux smartphones (Mobilinux 4.1) le noyau 2.6.10, ainsi que le support de la glibc. Il y apporte néanmoins quelques modifications et ajouts pour le rendre viable dans le monde des mobiles. Ainsi, MontaVista y ajoute un support avancé de la gestion d'énergie, un boot rapide (permettant de démarrer le mobile en moins de 10 secondes), le support temps réel avec un ordonnanceur temps réel, et une gestion plus fine de la mémoire. Il faut encore ajouter à cela le support de matériel spécifique présent dans certains appareils, comme le support pour la présence d'un appareil photo et toute la gestion de la téléphonie (GSM, GPRS, 3G, ...). L'interface graphique est aussi un module spécialisé et dans le cas de MontaVista, il utilise smallX, une trousse à outils graphiques (toolkit) basé sur l'implantation de X Windows System, GTK+, FreeType et IceWM.

Il faut néanmoins se rendre compte que les implantations de Linux dans les appareils mobiles peuvent parfois varier énormément d'un appareil à un autre. Il aurait été intéressant de connaître en détails les points communs ou

les différences entre les implantations de Linux pour les appareils mobiles et le monde PC, mais je n'ai pu trouver de documentations précises à ce sujet. Néanmoins si le lecteur est intéressé, l'article [58] explique brièvement le processus de réduction d'un système Linux ainsi que les étapes de la réduction de l'occupation mémoire et disque pour le rendre opérationnel sur un système embarqué.

Enfin j'aimerais souligner la présence récente d'une volonté de *standardisation* dans monde du système d'exploitation Linux pour appareils mobiles. Le forum LiPS (Linux Phone Standards, [59]) s'efforce d'établir un environnement de base commun pour le développement d'applications sur appareils mobiles et le groupe Mobile Linux Initiative (MLI) de l'Open Source Development Labs (OSDL, [60]) a pour mission de définir les spécifications et les capacités du noyau dans le même domaine. Peu à peu, divers constructeurs et sociétés de développement y adhèrent.

2.3.5 Windows CE

Windows CE est le système d'exploitation pour appareils mobiles de Microsoft. Windows CE existe en différentes versions et peut être personnalisé par les fabricants pour s'adapter à leurs besoins. Souvent, on confond Windows CE, Windows Mobile et Pocket PC et sont employés l'un pour l'autre. En pratique ce n'est pas tout à fait exact. Windows CE est un système d'exploitation modulaire qui sert de fondation pour de nombreux types d'appareils.

Windows Mobile peut être décrit comme un sous-ensemble de fonctionnalités de Windows CE. Pour le moment, Pocket PC (nommé aussi Windows Mobile pour Pocket PC), Smartphone et PocketPC Phone Edition sont les trois plateformes principales. Chacune de ces plateformes utilise différents composants de Windows CE, et rajoute des fonctionnalités supplémentaires dépendant de leurs appareils respectifs.

Windows CE est un système multi-tâches, mais comme il fut optimisé pour un environnement mobile, certaines fonctionnalités sont absentes, comme le support multi-utilisateurs. Windows CE essaie de réutiliser un certain nombre de concepts de la version bureautique de Windows dans le but d'inciter les utilisateurs et les développeurs à utiliser et développer des applications dans leur environnement familier. Similairement à Linux, Windows CE supporte une grande variété d'architectures matérielles (il supporte les architecture Intel x86, MIPS, ARM, PPC et Hitachi SuperH) et est donc utilisé par de nombreux fabricants.

L'architecture de Windows CE est relativement proche de celle de windows 95. Elle est composée de différentes couches d'abstraction matérielle permettant de rendre indépendant le matériel du logiciel. Cette couche d'abstraction matérielle est très importante pour les constructeurs de matériels, car elle permet de réduire considérablement les coûts de développement et surtout d'évolution du matériel : lorsque le matériel évolue, il suffit d'écrire un nouveau pilote pour ce matériel sans changer tout ce qui existe au dessus et au dessous.

2.3.6 BlackBerry

Le BlackBerry est un appareil sans fil commercialisé pour la première fois en 1999 qui supporte, outre la fonctionnalité de téléphone mobile, la navigation web, l'envoi et la réception d'email en temps réel et la fonctionnalité de fax via internet. Il fut développé par Research in Motion (RIM) et délivre l'information à travers les réseaux de données sans fil des compagnies de téléphone mobile.

Le BlackBerry pénétra le marché du fait qu'il se concentre avant tout sur les emails, et outre le fait qu'il implante des applications habituelles de PDA (agenda, carnets d'adresses, traitement de texte), il est avant tout un véritable terminal autonome pour l'envoi et la réception d'emails, grâce notamment à son habilité à s'appuyer sur un réseau sans fil particulier offert par certaines compagnies de téléphone.

Les appareils BlackBerry sont par contre dépendant de la couverture réseau de l'opérateur téléphonique et n'implante que très récemment (mi 2006) un support Wi-fi.

Ces appareils sont surtout très populaires dans certaines entreprises où ils sont utilisés principalement comme terminaux d'emails. En effet, pour intégrer complètement le BlackBerry dans une compagnie, l'installation de serveurs spécifiques, les BlackBerry Enterprise Server (BES) est souhaitable. L'utilisateur est alors averti en temps réel de la réception d'un email, celui-ci étant directement transféré du BES (qui est en relation directe avec un serveur d'email, souvent Microsoft Exchange Server ou Domino) sur son BlackBerry en utilisant le réseau cellulaire si celui-ci est connecté. La figure 2.1 illustre cet échange.

Le BES fournit également une connectivité TCP/IP pour les BlackBerry qui y sont connectés, dont un composant, le Mobile Data Service (MDS) qui officie en tant que proxy. Cela permet à des applications développées par exemple en Java d'accéder à Internet. De plus, le BES fournit un certain

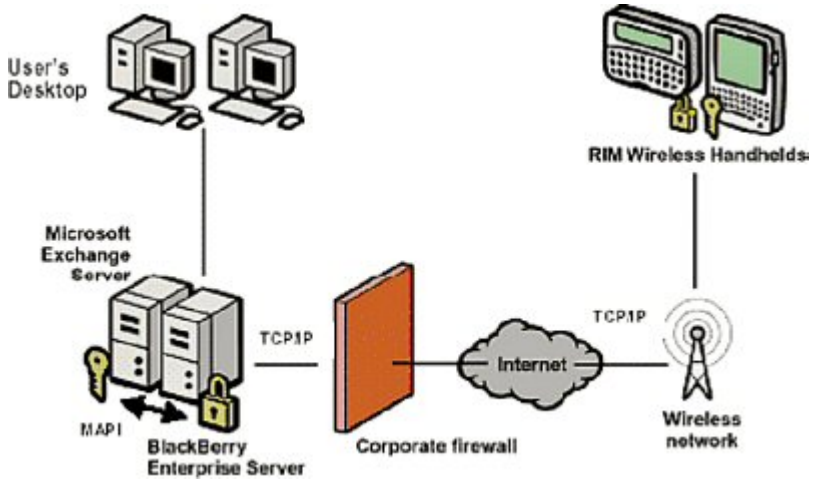

FIG. 2.1 – Architecture réseau du BlackBerry.

niveau de sécurité, en cryptant toutes les données qui sont communiquées entre le BlackBerry et le serveur, auparavant en utilisant l'algorithme triple DES, plus récemment en utilisant AES.

2.4 Logiciel pour Mobiles

Il existe certaines différences générales entre les applications qui sont exécutées par un appareil mobile et les applications développées pour un ordinateur de bureau. La plupart de ces différences sont dues à l'interface utilisateur assez limitée sur les appareils mobiles. Dans le but de fournir une meilleure connaissance de la nécessité de logiciels spéciaux pour appareils mobiles, une interface utilisateur typique sera décrite en détails dans cette section. Cependant examinons d'abord le matériel.

Comme mentionné dans la section 2.1.1 ci-avant, les appareils mobiles ont généralement un très petit affichage, un clavier réduit, et la souris est remplacée par un écran tactile. De ce fait, presque toutes les interfaces pour appareils mobiles sont développées afin de supporter seulement une seule application interactive à la fois. Certains systèmes n'autorisent strictement qu'une seule application qui peut être exécutée à la fois, et force donc la terminaison de l'application courante avant de pouvoir en lancer une nouvelle.

L'interface utilisateur limitée et les possibilités d'entrée ont aussi un grand impact sur les applications. Par exemple, la plupart des applications sont

développées pour une entrée de texte minimale, et souvent affiche une boîte de sélection à la place d'un champ de texte. D'autres différences sont dues au faible espace de stockage disponible, et à la faible vitesse des processeurs.

En général, les applications peuvent être classées en deux groupes. Soit elles ressemblent plus à une version allégée d'une application existante sur ordinateur personnel (comme un traitement de texte ou un navigateur internet), soit elles sont développées spécifiquement pour être utilisées sur un appareil mobile. Ce développement spécifique est le sujet de la section suivante.

2.4.1 Applications spécifiques pour appareils mobiles

Il y a de nombreuses applications que l'on trouve exclusivement sur un appareil mobile, comme une application qui détermine les paramètres d'un téléphone portable (dans le cas où l'appareil possède des fonctionnalités de téléphone portable). Cette section fournit un bref aperçu de telles applications.

L'application la plus importante est celle de synchronisation. Comme son nom l'indique, cette application est utilisée pour synchroniser les données stockées sur un appareil mobile avec celles stockées sur un ordinateur personnel ou un serveur. De plus, ces applications sont aussi utilisées pour installer des logiciels supplémentaires ou modifier automatiquement les paramètres systèmes (l'horloge système par exemple). Souvent, cette synchronisation permet aussi l'accès à d'autres fonctionnalités, comme l'accès au système de fichier de l'appareil mobile, et des facilités de déboguage à distance.

Une autre application se charge de contrôler les différentes interfaces sans fil. Cette application est utilisée pour configurer les interfaces et les services qui leur sont liés. Alors que ce type d'application est courant dans les environnements réseaux, les versions pour appareils mobiles supportent généralement des paramètres de configuration dépendant de l'heure et un *routage* basé sur le coût associé à l'interface (que l'on peux aussi remplacer par la quantité de bande passante disponible).

D'autres applications fournissent une interface pour du matériel embarqué supplémentaire, comme un téléphone portable, une caméra numérique ou un navigateur GPS. Souvent ces applications fournissent une interface pour d'autres applications tierces.

2.4.2 Développement de logiciels pour appareils mobiles

Les appareils mobiles ont plusieurs aspects qui rendent le développement plus difficile. Cette section fournit un aperçu du sujet et souligne certains points importants.

Quand on développe un logiciel pour appareils mobiles, la difficulté principale est la présence de différentes architectures matérielles utilisées dans ces appareils. Dans la plupart des cas, le développeur a besoin d'effecteur du *cross-developpement* et du *cross-testing*. Ceci peut nécessiter de devoir développer un logiciel sur une plateforme matérielle non native et un système d'exploitation non natif, et ensuite de tester ce logiciel dans un environnement émulé.

L'émulation est principalement utilisée pour accélérer le processus de développement car il supprime le besoin de transférer l'application vers un appareil chaque fois qu'on veut l'exécuter. De plus, la plupart des appareils ne proposent pas d'interface de déboguage décente, ce qui peut aussi obliger les développeurs à utiliser des émulateurs pour tester et déboguer leur programme. D'un autre côté, les applications qui dépendent d'un matériel ou service spécial, comme le Bluetooth ou un réseau téléphone cellulaire, demande un matériel réel pour être testé, ce qui implique un besoin plus important en temps de développement. De plus, l'émulation peut poser quelques inconvénients (un processeur émulé peut être plus lent ou plus rapide que dans la réalité), et ne peut donc être considéré que comme un complément au test réel.

Un autre point important est la diversité entre les différents appareils mobiles. Cela va des différences de matériels ou de systèmes d'exploitation, aux différentes résolutions d'écran ou des fonctionnalités manquantes du système d'exploitation. Cette diversité de versions implique souvent aux développeurs d'inclure la possibilité de vérifier la présence de fonctionnalités dans leurs applications dans le but de fournir une portabilité entre différentes versions d'un appareil.

De plus sur nos ordinateurs de bureau, nous avons l'habitude de télécharger des patches pour mettre à jour nos applications ou notre système d'exploitation, pour corriger des bogues et résoudre des problèmes de sécurité. Mais comment sont gérées ces mises à jour sur les appareils mobiles ? Pour faire court, ça ne l'est pas. Les fabricants de téléphones portables produisent de nouvelles versions de leur logiciel pour téléphones mobiles, mais elles n'atteignent généralement pas l'utilisateur qui a déjà acheté un appareil. La plupart des téléphones mobiles doivent être réexpédiés en magasin

(en réparation) pour espérer avoir une mise à jour de leur système. Seuls quelques utilisateurs avertis sont capables de mettre à jour le système de leur téléphone mobile, mais pour la majorité des utilisateurs les bogues présent lors de l'achat de leur mobile resteront pendant toute la durée de vie de leur appareil.

La nécessité de mieux tester les logiciels créés est aussi à prendre en considération. Beaucoup de développeurs ne testent leurs applications que sur quelques appareils (typiquement 4-5) présents sur le marché. Or l'implantation des API ou framework communs à une majorité d'appareils mobiles peuvent avoir des différences d'implantation ou des bogues d'un appareil à un autre. Pour être réellement sûr que l'application fonctionnera, les tests doivent portés sur un plus grand nombre d'appareils mobiles, en testant par exemple plusieurs modèles de fabricants différents. Bien sûr, tester un grand nombre d'appareils (40-50) a un coût et prend du temps. Il faut donc considérer prudemment les ressources dépensées par la phase de tests, pour ne pas prendre le risque de fournir une application aux utilisateurs n'ayant pas été testée sur leur appareil.

En conclusion, le développement de logiciel pour un appareil mobile est plus complexe et prend plus de temps que le développement d'un logiciel pour un ordinateur personnel.

Chapitre 3

Sécurité des appareils mobiles

Les problèmes liés à la sécurité des appareils mobiles sont différents des problèmes de sécurité des ordinateurs personnels ou des serveurs. Comprendre ces différences est important dans le but d'appréhender la sécurité dans les appareils mobiles. Ce chapitre fournit un résumé des problèmes de sécurité dans ces divers appareils.

3.1 Compréhension

La sécurité des appareils mobiles se distingue de la sécurité informatique conventionnelle sur plusieurs aspects : la mobilité, les fonctionnalités réduites, la convergence technologique, une forte connectivité et une forte personnalisation.

Mobilité

Les appareils mobiles sont mobiles ;) Ils ne sont pas gardés dans un endroit sécurisé, et peuvent donc être volés ou manipulés.

Fonctionnalités réduites

Les appareils mobiles sont des ordinateurs mais il leur manque un certain nombre de fonctionnalités que possède un ordinateur de bureau. Par exemple, un appareil mobile ne possède pas un clavier entier et n'a qu'une possibilité de traitement limitée.

Convergence technologique

Les appareils mobiles d'aujourd'hui combinent énormément de technologies différentes en un seul engin, comme un PDA, un téléphone mobile, un baladeur ou un appareil photo numérique.

Forte connectivité

Beaucoup d'appareils supportent de multiples façons de se connecter à Internet ou à tout autre réseau.

Forte personnalisation

Les appareils mobiles ne sont généralement pas partagés entre les utilisateurs, là où les ordinateurs le sont souvent. Ces appareils ne sont jamais loin de leur propriétaire.

En regroupant tous ces aspects, on peut voir la raison pour laquelle la sécurité des appareils mobiles est plus complexe que celle des ordinateurs normaux. La mobilité, par exemple, augmente le risque de vol de données, car voler un appareil mobile est beaucoup plus simple que de pénétrer dans un ordinateur personnel. Une forte personnalisation, couplée avec une forte connectivité, augmente la menace de violation de la vie privée (un appareil mobile se situe là où se trouve son propriétaire, et donc localiser le mobile signifie localiser son propriétaire). La convergence technologique conduit à plus de risques de sécurité. Chaque nouvelle fonctionnalité ajoute au moins une nouvelle cible qui peut être potentiellement attaquée. Les fonctionnalités réduites peuvent faciliter certaines attaques par Déni de services (Denial-of-Service attacks - par exemple des attaques dont le but est de rendre l'appareil temporairement inutilisable). De plus, le manque de fonctionnalité, comme l'absence d'un clavier complet, complique l'implantation de mécanisme d'authentification (traditionnellement le couple nom d'utilisateur / mot de passe). Tous ces aspects apportent de nouvelles implications, comme l'augmentation de la complexité lors d'audits de sécurité sur les appareils mobiles.

3.2 Menaces

3.2.1 Perte ou vol d'appareil

Si un appareil est perdu ou volé, sa confidentialité est cassée. Son intégrité peut être endommagée si l'appareil réapparaît après une période de temps. Dans ce cas, quelqu'un peut avoir installé un logiciel espion (spyware) ou avoir causé une détérioration matérielle qui peut avoir abîmé l'intégrité de l'appareil. Pendant son absence l'appareil n'est, bien entendu, plus disponible pour son propriétaire bien qu'il soit probable qu'un dispositif critique soit remplacé rapidement après qu'il vient à manquer.

Perdre ou se faire voler un appareil n'est pas spécifique aux appareils mobiles, cela se produit aussi pour les ordinateurs portables et autre matériel informatique, comme des disques durs ou clés USB. Cependant, les appareils mobiles sont plus susceptibles de disparaître car ils sont petits et constamment transportés par leurs utilisateurs. De plus, comme dit plus haut, il n'existe généralement pas d'identification sur ces appareils (ou ceci n'est pas activé par défaut), ce qui facilité encore la pénétration de ces systèmes.

3.2.2 Attaques par déni de service

Les attaques par déni de service (Denial-of-Service, DoS) existent depuis très longtemps et ne sont pas nouvelles ni spécifiques aux appareils mobiles. Les attaques DoS ont pour but de rendre un service ou un appareil inutilisable pour son utilisateur, en le rendant indisponible.

Les problèmes des attaques DoS contre les appareils mobiles sont imputables principalement à leur forte connectivité et fonctionnalités réduites. Par exemple, une attaque DoS courante consiste à envoyer une grande quantité de trafic à un hôte connecté au réseau. Alors qu'un attaquant a besoin de beaucoup de ressources pour attaquer un ordinateur normal ou un serveur, un appareil mobile, par le fait de sa capacité de traitement limitée, peut être plus facilement rendu inutilisable par l'envoi massif de trafic depuis l'attaquant.

D'autres attaques DoS plus spécifiques contre des appareils mobiles peuvent utiliser le fait que ces appareils tournent sous batteries. Dans ce cas, le but de l'attaque est de décharger le plus vite possible les batteries de la cible. Des attaques réussies pourraient alors arrêter ou fortement limiter le temps de fonctionnement de la cible. De telles attaques peuvent exploiter plusieurs fonctionnalités, comme des tâches CPU intensives nécessitant beaucoup d'énergie

(par exemple, des routines de chiffrement complexes, comme la négociation SSL) ou forçant l'appareil à rallumer certaines parties du matériel qui étaient éteintes (comme par exemple l'affichage - voir [43]).

Enfin des attaques comme le brouillage d'une bande de fréquence entière (par exemple la bande des 2,4 GHz, utilisée pour le Wi-Fi et le Bluetooth) est aussi très efficace.

3.2.3 Attaques par réseau sans fil

Il existe un grand nombre d'attaques différentes qui influencent la connectivité d'une cible. La plus commune est l'écoute des transmissions sans fil pour en extraire des informations confidentielles, comme les noms d'utilisateurs et mots de passe. Les attaques par *écoute* ne sont pas des attaques spécifiques aux appareils mobiles, mais ces derniers sont particulièrement vulnérables du fait que bien souvent ils ne supportent que la communication via une connexion sans fil.

Une autre forme d'attaques sans fil usurpe l'unique identification matérielle (comme l'adresse MAC, Medium Access Control) présente dans toutes les communications sans fil, dans le but de traquer ou d'identifier le propriétaire de l'appareil. Ce type d'attaque influence la forte personnalisation des appareils mobiles et est spécifique à ceux-ci.

3.2.4 Attaque par effraction

C'est une attaque où l'attaquant parvient à prendre le contrôle partiel ou total de la cible. Les attaques par effraction (Break-In attacks) existent en deux catégories : injection de code et exploitation d'erreurs logiques. L'injection de code est obtenue généralement en exploitant des erreurs de programmation qui mènent à des dépassements de tampon (buffer overflow) ou des vulnérabilités de chaînes de caractères. Les abus d'erreurs logiques sont plus subtils, car une erreur logique particulière est très dépendante de l'application ou de l'appareil attaqué.

Ces attaques ont un impact sur l'intégrité, la confidentialité et la disponibilité des appareils. La menace réelle de ces attaques dépend grandement du but de l'attaquant. Généralement elles préparent le terrain pour d'autres types d'attaques, comme le vol d'identité ou de données.

Une attaque par dépassement de tampon et injection de code sur un système Windows CE sera présentée en détails dans le chapitre 5.

3.2.5 Virus, vers et chevaux de Troie

Les virus, vers et chevaux de Troie sont des menaces pour les appareils mobiles, de la même manière qu'ils le sont pour les ordinateurs normaux. Ils volent ou détruisent les données et peuvent rendre les systèmes infectés inutilisables.

Les vers qui ciblent des smartphones peuvent aussi avoir un coût s'ils se répandent en utilisant un service pour lequel l'utilisateur est facturé, comme le MMS par exemple. Dans ce cas, un vers s'envoyant lui-même à des centaines de téléphones mobiles peut causer un dommage substantiel au propriétaire de l'appareil infecté.

Les virus qui ciblent aussi bien les appareils mobiles que les ordinateurs sont aussi possibles. Ces virus peuvent initialement infecter un appareil mobile (peut être en utilisant une connexion sans fil) et peuvent par après contaminer un ordinateur de bureau (après une synchronisation par exemple). Ce type de virus ou de vers peut facilement outrepasser les mécanismes de sécurité configurés seulement pour détecter des attaques externes.

Les virus et vers peuvent aussi placer un cheval de Troie sur l'appareil, permettant le vol des données ou l'enregistrement des activités téléphoniques d'un utilisateur, en envoyant périodiquement un rapport (par exemple à chaque fois que l'utilisateur se connecte à Internet en utilisant le GPRS).

3.2.6 Attaques basées sur les infrastructures

L'infrastructure de service, qui est construite sur base de réseaux GSM et de serveur d'applications, joue un rôle important dans le monde des appareils mobiles. Elle représente la base des fonctionnalités des appareils mobiles, comme la fonctionnalité téléphonique ou de suivi d'emails. Alors que les appareils peuvent être sécurisés jusqu'à un certain degré, l'infrastructure doit être comparativement ouverte dans le but d'être utilisable. Et donc, une attaque basée sur l'infrastructure peut être ciblée vers de multiples autres appareils sécurisés. Bien que ces attaques puissent être classifiées dans une des catégories mentionnées plus haut (DoS ou attaque par réseau sans fil), elles doivent être mentionnées explicitement à cause de leur interaction avec l'infrastructure mobile.

3.2.7 Attaques par surfacturation

Une attaque par surfacturation (overcharging attacks) est une attaque qui implique un service payant donné, comme par exemple un service téléphonique mobile. Le but ici est de charger le compte de la victime de frais supplémentaires, et, si possible, de transférer ces frais supplémentaires de la victime à l'attaquant.

Un exemple de ce type d'attaque est décrit dans [65]. Dans ce cas spécifique, l'attaquant profite d'un défaut dans le système GPRS pour facturer les autres clients du même fournisseur téléphonique. Cette attaque utilise le fait que le mobile est connecté en permanence au réseau GPRS (la facturation se fait en fonction du volume de données transférées). La seule chose dont l'attaquant a besoin de faire est d'envoyer un trafic quelconque vers l'adresse IP de la victime. Le fournisseur ne vérifie pas si le trafic a été demandé par la victime, et facture celle-ci.

Ce type d'attaque est très spécifique des appareils mobiles, car beaucoup de services sans fil sont facturés suivant le volume de données transférées sur base de l'adresse de l'appareil ou d'un identifiant.

Chapitre 4

Plateforme Java 2 Micro Edition

Comme nous l'avons déjà vu, un smartphone est un téléphone évolué intégrant aujourd'hui un large écran couleur et possédant une plus grande puissance de traitement qu'un téléphone mobile classique. Une des fonctionnalités clé est le fait de pouvoir installer des programmes additionnels. Le marché des smartphones évolue rapidement. La disponibilité toujours plus grande de ces appareils stimule le marché à produire de nombreux contenus, comme des jeux, des applications multimédia (audio/vidéo) ou même du contenu « pour adulte ».

Beaucoup de plateformes de développement différentes existent pour les smartphones, généralement classées suivant le fabricant de l'appareil, le système d'exploitation utilisé et les fonctionnalités du mobile. La plateforme la plus répandue est Java 2, Micro Edition (J2ME), disponible sur plus de 80% des smartphones actuellement sur le marché.

Ce chapitre commencera par présenter la plateforme J2ME, quelles sont les technologies disponibles et comment elles sont implémentées dans les appareils mobiles, quelles sont les différences avec les autres implantations de Java, quels sont les mécanismes de sécurité courants et les vulnérabilités existantes.

Ce chapitre est basé sur les travaux de [52, 53, 54, 55, 56, 71]. Les illustrations de code seront tirés en majeure partie du [71].

4.1 Brève introduction à Java

La technologie Java, introduite en 1996 par Sun Microsystems, est devenue depuis lors l'une des techniques de programmation les plus répandues

dans le monde. Cette technologie a initialement été créée pour faire face à deux problèmes importants : la portabilité des programmes et la facilité du développement logiciel. En effet, les programmes Java, une fois écrits, peuvent être exécutés sur toute machine *d'une même classe* (il faut comprendre par là d'un même type de machine virtuelle), peu importe les différences architecturales présentes (par exemple, sur un processeur Intel ou Motorola) ou le système d'exploitation utilisé (par exemple, Linux ou Windows).

Cette compatibilité entre plateformes est obtenue grâce au concept de *Machine Virtuelle Java* (Java Virtual Machine, JVM), qui consiste essentiellement en un logiciel qui émule un processeur abstrait, avec son propre ensemble d'instructions, sur une plateforme logicielle et matérielle donnée. Les programmes Java sont alors précompilés suivant cet ensemble d'instructions ; à l'exécution, la machine virtuelle traduit ce code précompilé en séquences d'instructions compréhensibles par le processeur de la machine exécutant le code, maintenant le programme non conscient des spécificités propre à l'environnement d'exécution courant.

Avec le concept de machine virtuelle, Sun introduit le langage de programmation Java, dans le but de programmer la JVM. Le (langage de programmation) Java est orienté objet, et est basé sur l'utilisation de classes polyvalentes. Sa syntaxe est similaire au langage de programmation bien connu qu'est le C/C++, cependant beaucoup de constructions non sécurisées ou sensibles ont été enlevées. Cette simplicité du langage permettant un développement facile, couplée avec une grande portabilité, ont encouragé de nombreux déploiements à travers le monde.

Encouragé par le succès de sa technologie, Sun décida de s'étendre dans de nouveaux domaines, les serveurs de programmation d'entreprise d'un côté, et les petits appareils à ressources limitées de l'autre. Java s'est alors divisé en trois éditions, la technologie traditionnelle prenant le nom de Java 2 Standard Edition (J2SE) ; son sur-ensemble, étendu avec des fonctionnalités d'entreprise comme l'accès aux bases de données et la génération dynamique de contenu web fut appelé Java 2 Enterprise Edition (J2EE), alors que son sous-ensemble, conçu pour de petits appareils aux capacités limitées, fut nommé Java 2 Micro Edition (J2ME). La figure 4.1 illustre ces différentes versions.

4.2 Description de la plateforme J2ME

Alors que les ordinateurs personnels et serveurs d'entreprises ont des capacités similaires dans leur classe respective, il faut remarquer qu'il existe une

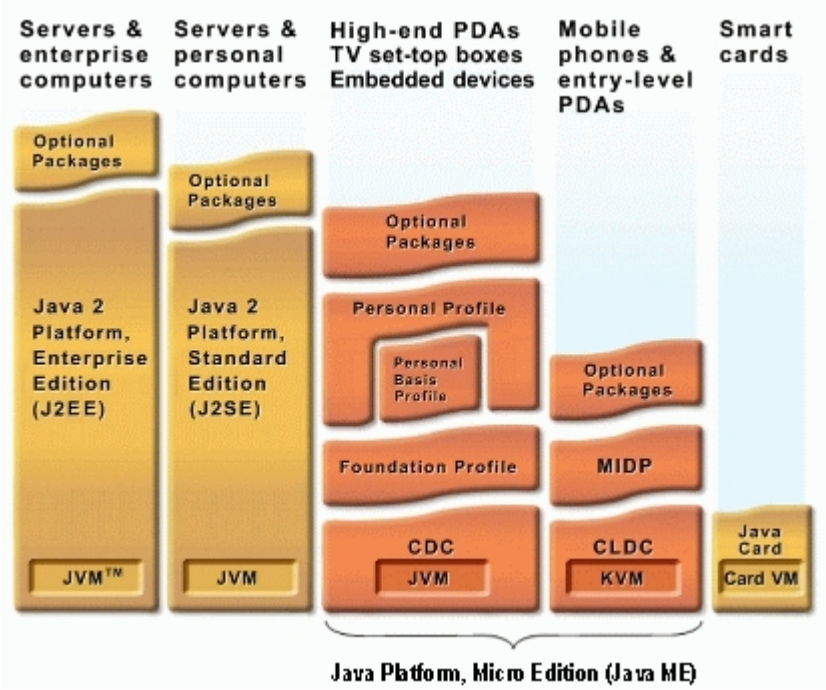

FIG. 4.1 – La plateforme Java.

grande variété de types de petits appareils, possédant des buts et des propriétés variés. Pour cette raison, J2ME n'est pas une technologie consistante, mais plutôt une collection de différentes *configurations* de machines virtuelles et de bibliothèques (nommées *profile*) construites sur ces configurations. Les *configurations* et les *profils* définissent précisément toutes les fonctionnalités du langage qu'une personne peut utiliser pour programmer un certain type d'appareil.

4.2.1 Machine virtuelle Java - KVM

La machine virtuelle Kilo (Kilo Virtual Machine, KVM) est l'implantation de Sun de la machine virtuelle Java Micro Edition avec la configuration CLDC (le terme raccourci Java ME CLDC sera utilisé par la suite pour la désigner). Elle est dédiée aux appareils supportant la configuration CLDC, et celle-ci est présente dans l'implantation Java supportée par les smartphones du marché. Dans la suite de ce chapitre, c'est cette machine virtuelle là qui sera analysée. KVM est le résultat de la volonté de Sun de produire une machine virtuelle de petite taille pouvant supporter les fonctionnalités

du langage Java dans un environnement restreint (limitations de mémoire, de puissance du processeur et d'énergie). Celle-ci pèse entre 50 et 70 KB, avec une taille totale d'environ 128 KB en mémoire, en incluant la machine virtuelle, les bibliothèques Java minimales spécifiées par la configuration, et un peu d'espace pour le tas (heap) requis pour exécuter une application. Son principal rôle est l'exécution des applications Java. Ces applications doivent être installées sur l'appareil avant leur exécution, et le processus d'installation requière généralement une connection réseau. KVM consiste en un nombre de modules fonctionnels, principalement le module de démarrage, le chargeur, le vérificateur, l'interpréteur, l'interface native, le ramasse-miettes (garbage collector), le gestionnaire de *thread* et l'antémémoire en ligne (cache inline).

4.2.2 Configuration - CLDC

Les configurations définissent des classes d'appareils possédant des caractéristiques similaires du point de vue mémoire et processeur. Les profils, d'un autre côté, définissent des appareils similaires en terme de type d'écran, de possibilités d'entrées et de connectivité réseau, et sont un complément aux fonctionnalités bas niveaux des configurations en ajoutant le support pour, par exemple, l'interface utilisateur ou la connectivité réseau.

Une configuration spécifie les fonctionnalités que doit supporter la machine virtuelle Java, les fonctions du langage de programmation Java à inclure, et les bibliothèques Java et API (Application Programming Interface) supportées. Deux configurations sont largement répandues sur les appareils mobiles supportant J2ME, à savoir Connected Limited Device Configuration (CLDC) version 1.0 et 1.1. La version 1.1 est disponible dans les nouveaux appareils mobiles, alors que la version 1.0 existent depuis des années. La différence principale entre les deux est que CLDC 1.1 supporte les opérations en virgules flottantes. CLDC 1.0 est donc vu comme un sous-ensemble de CLDC 1.1, les applications compilées pour CLDC 1.0 continuant à tourner sur la version 1.1.

4.2.3 Profils - MIDP

Deux profils existent, qui étendent les fonctionnalités de CLDC, à savoir Mobile Information Device Profile (MIDP) version 1.0 et 2.0. Les profils MIDP ajoutent les API pour l'interface utilisateur, la connectivité réseau et le stockage de données. La connectivité réseau est fournie à travers des connexions HTTP. De nouveau, MIDP 1.0 est un sous-ensemble de MIDP 2.0,

et les différences principales sont le support dans MIDP 2.0 des connexions sécurisées HTTPS ainsi qu'une API graphique plus puissante pour les jeux.

La figure 4.2 illustre l'architecture en couches vue dans cette section.

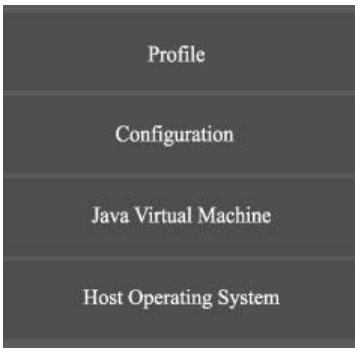

FIG. 4.2 – Architecture en couches.

4.2.4 MIDlets

Un *MIDlet* est une application MIDP, qui est un programme Java destiné à être exécuté sur un appareil conforme MIDP. Ce nom est une continuation des modèles de noms *Applet*, *Xlet*, *Servlet*, ...

Un MIDlet est bien sûr écrit dans le langage de programmation Java. Cependant, les API sont différentes de celles utilisées en Java SE. Un programmeur Java ME est restreint à utiliser les API CLDC, les API MIDP, et éventuellement des packages optionnels. La classe *javax.microedition.midlet.MIDlet* est la classe utilisée pour représenter un MIDlet. Un MIDlet possède typiquement les méthodes *startApp()*, *pauseApp()* et *destroyApp()* (le cycle de vie d'un MIDlet sera expliqué plus en détail par la suite). Ces méthodes permettent au logiciel de gestion des applications (Application Management Software, AMS) de l'appareil de gérer et synchroniser les activités des MIDlets (les créer, démarrer, mettre en pause ou détruire). Le programmeur Java ME peut définir la classe MIDlet comme implémentant l'interface *CommandListener* permettant de recevoir et de traiter des événements hauts niveaux, comme par exemple la pression d'un bouton de l'appareil.

4.2.5 Prévérification

Une nouvelle étape dans le développement d'une application Java ME (MIDlet) est l'étape de *prévérification*. Une telle étape n'existe pas lorsque

l'on veut construire un programme Java SE. En effet, étant donné les contraintes de ressources limitées, l'étape classique de vérification des fichiers de classes (qui consiste à vérifier l'accès à la mémoire en dehors de son espace d'exécution et le remplacement d'une classe de java.* ou javax.*) est divisée en deux étapes : une étape après la compilation, appelée prévérification et une étape avant l'exécution, appelée vérification.

Avant le déploiement de l'application, le développeur emploie l'utilitaire de prévérification qui modifie le fichier *class* par ajout de bytecode indiquant que la classe est correcte. Au cours de l'exécution, la KVM vérifie la présence de ce flag et l'exactitude de son information. En cas de problème, le chargement de la classe est interrompu et une exception est levée. Ceci permet de ne pas procéder à toute la vérification des classes juste avant l'exécution, étape coûteuse en temps et puissance de calcul.

Je n'ai par contre pas pu trouver de documentations expliquant s'il existe un quelconque mécanisme pour sécuriser le fichier *class* prévérifié (signature du code par exemple) dans le but de garantir son intégrité.

4.3 Sécurité dans Java ME

Java a été conçu en mettant l'accent sur les aspects sécuritaires. Les principaux aspects sécuritaires sont les suivants :

- Le langage Java est fortement typé, permettant l'élimination de bogues de programmation et le respect de la sémantique du langage vérifié à la compilation. Le résultat de la compilation d'un programme Java est un code intermédiaire appelé *bytecode*.
- Une application Java compilée ne peut être exécutée avant que le vérificateur de bytecode ne prouve que l'application a bien été compilée en suivant les règles de sécurité de Java.
- Chaque classe Java est chargée avec sûreté grâce au chargeur de classe (classloader) qui est responsable de trouver les classes, les charger depuis leur emplacement et appeler le vérificateur pour vérifier le bytecode chargé avant son utilisation.
- Une politique sécuritaire définit les règles nécessaires assurant un accès sûr aux ressources sensibles par les applications Java. Ces règles sont définies en termes de permissions accordées aux classes Java suivant leur degré de fiabilité (trust) dans le système.
- Le gestionnaire de sécurité (Security Manager) est le mécanisme appliquant cette politique de sécurité. Il s'assure que toutes les règles définies

par la politique de sécurité sont respectées par toute application s'exécutant sur la plateforme. Il intercepte les appels sensibles (du point de vue de la sécurité) et passe les permissions de l'appel et l'action sous contrôle au contrôleur d'accès (Access Controller).
- Le contrôleur d'accès décide alors si le code appelé possède les permissions nécessaires pour exécuter l'action sensible ou non ; si oui, l'action est exécutée ; sinon l'action est refusée et une exception est lancée.
- Le code des applications Java peut être composé de plusieurs composants distribués sur différentes machines. La communication entre ces différents composants situés sur plusieurs machines est habituellement effectuée via des réseaux publics. Java doit donc assurer une sécurité point à point en permettant l'authentification et l'autorisation de composants et la mise en place de canaux cryptés de communication.

4.3.1 Le modèle du bac à sable

La base de la sécurité de Java repose sur le modèle du *bac à sable* ou *Sandbox*. Le bac à sable est simplement une zone de mémoire limitée dans laquelle du code non fiable (typiquement téléchargé depuis un réseau) peut être exécuté sans pouvoir effectuer une quelconque action dangereuse. Au commencement (JDK 1.0), les programmes Java étaient classés de fiables ou non fiables (*trusted* ou *untrusted*). Un code fiable consistait en un code d'une API Java, alors qu'un code téléchargé depuis un réseau était jugé non fiable. Avec le JDK 1.1, le modèle du bac à sable fut étendu, autorisant certains codes téléchargés à être considérés comme fiables. Ceci est rendu possible par l'utilisation de signatures digitales. Une liste de clés publiques est alors maintenue sur la plateforme, où chaque clé est liée à une identité via une autorité de certification. Depuis le SDK 1.2, le bac à sable Java est encore étendu pour que plusieurs niveaux de sécurité puissent être spécifiés par la plateforme Java. Néanmoins, l'utilisateur final possède toujours le dernier mot, pouvant choisir explicitement de lancer son application dans le bac à sable ou non, suivant son degré de confiance en l'application. La figure 4.3 illustre le modèle sandbox.

4.3.2 Contraintes dues à ME

Le modèle conventionnel de sécurité de Java est considéré comme non approprié pour la plateforme Java ME CLDC. En effet, le volume total de

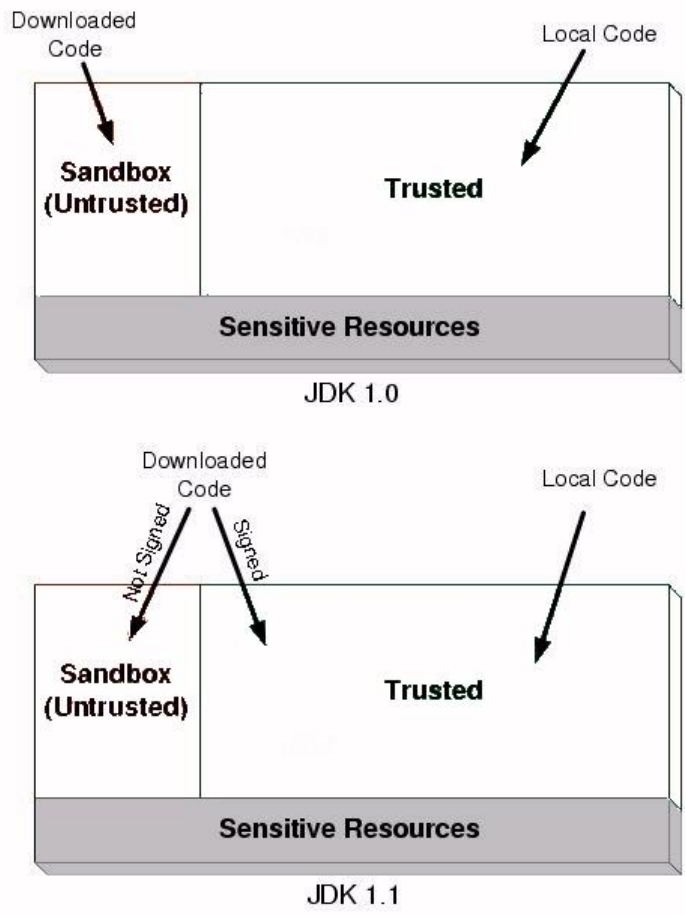

FIG. 4.3 – Le modèle sandbox.

code dédié à la sécurité dans Java SE excède de loin la capacité mémoire totale disponible pour CLDC. Par conséquent, ce modèle conventionnel a été remplacé par un modèle de sécurité allégé, effectuant une grande partie de la vérification du bytecode en dehors de l'appareil (ceci a déjà été discuté au point 4.2.5), adoptant un modèle de bac à sable simplifié et fournissant des mécanismes de sécurité point à point allégés.

Cependant, certaines fonctionnalités non présentes dans J2ME sont bénéfiques du point de vue sécuritaire, car celles-ci peuvent justement poser un problème de sécurité. Ces fonctionnalités non présentes sont :

– *Java Native Interface* (JNI) : principalement pour des raisons de sécurité et de performance, JNI n'est pas implémenté dans CLDC. Bien qu'une interface native Kilo (Kilo Native Interface, KNI) est présente dans Java ME CLDC, KNI ne peut pas appeler et charger arbitraire-

ment des fonctions natives depuis des programmes Java. Cette absence protège la plateforme du téléchargement de code natif, contre lequel il est très difficile de se protéger, et qui peut causer une faiblesse de sécurité.
- *User-defined class loader* : Principalement pour des raisons de sécurité, le chargeur de classe est ici un composant intégré qui ne peut être surchargé, remplacé ou reconfiguré. Avoir un chargeur de classes unique protège la plateforme des attaques basées sur un chargeur de classe défini par l'utilisateur, qui sont une des principales techniques d'attaques sur les plateformes Java.
- *Thread group* : alors que CLDC supporte le *multi-threading*, CLDC n'inclut pas le support pour les groupes de *thread*.
- *Support for reflection* : aucune fonctionnalité de *reflection* n'est supportée (la possibilité pour le programme d'observer et de pouvoir changer sa sémantique, sa syntaxe ou son implantation durant l'exécution), et par conséquent il n'y a pas de support pour l'invocation de méthode distante (Remote Method Invocation). Ceci protège la plateforme à l'exposition de son architecture interne.

De plus, dans CLDC, le programmeur ne peut ajouter, surcharger ou modifier aucune des classes dans l'ensemble du système protégé, c'est-à-dire les configurations spécifiques, les profils spécifiques ou les spécificités du constructeur.

Enfin, une dernière restriction importante est qu'une application Java ne peut charger des classes que depuis son propre fichier d'archive Java (JAR). Par conséquent, les MIDlets d'un appareil ne peuvent interférer les uns les autres ou se voler des données et une application tierce ne peut accéder aux composants privés ou protégés des classes Java qui sont fournies comme applications système par le fabriquant de l'appareil.

4.4 Analyse de sécurité de Java ME CLDC

La section précédente présentait les concepts de sécurité de Java ME CLDC. Dans cette section, une évaluation de la sécurité de cette plateforme Java ainsi qu'une liste des vulnérabilités possibles dans l'implantation de cette plateforme seront présentées. La première section (Vulnérabilités reportées) présentera des vulnérabilités connues, alors que la suite ne développera que des pistes de vulnérabilités possibles, présentées dans des articles ou analyses ; toutes n'ont pas encore été exploitées à notre connaissance.

4.4.1 Vulnérabilités reportées

La base de données des bogues de Sun contient pas mal de références à propos de Java ME CLDC. Cependant peu de vulnérabilités affectant la sécurité dans Java ME CLDC y ont été reportées. Néanmoins un exemple est une vulnérabilité dans le SMS du Siemens S55. A côté de cela, quelques problèmes dans l'implantation de référence MIDP de Sun ont été reportés.

Vulnérabilité SMS du Siemens S55

En 2003, un groupe de pirates a découvert que le téléphone Siemens S55 possédait une vulnérabilité permettant d'envoyer des SMS depuis l'appareil sans l'autorisation de l'utilisateur. Cette attaque peut être menée grâce à un MIDlet malveillant qui, une fois chargé par la cible, envoie un message SMS sans demander de permission. Ceci était dû à une anomalie permettant au code Java d'outrepasser la requête normale de permission avec un affichage à l'écran arbitraire. L'utilisateur donnait alors sa permission pour l'émission du SMS, pensant qu'il répondait à une autre question affichée.

Problèmes dans l'implantation de référence MIDP de Sun

Créer un socket de connexion (`socket://hostname:portnumber`) nécessite les permissions nécessaires. Mais si quelqu'un exécute une connexion vers un port déjà utilisé, l'application ne vérifie pas les permissions. À la place, une exception est affichée (ConnectionNotFoundException), ce qui n'est pas correct car il n'y a aucune raison d'effectuer une connection si l'application n'en a pas le droit.

Un autre problème est le fait que le schéma basique d'authentification n'est pas entièrement supporté. Selon les spécifications MIDP 2.0, l'appareil doit être capable *de répondre à une réponse d'une requête HTTP 401 (Non autorisé) ou 407 (Authentification proxy requise) en demandant à l'utilisateur un nom d'utilisateur et un mot de passe et de renvoyer une requête HTTP avec ces informations. L'appareil DOIT être capable de supporter au moins le schéma d'authentification basic (RFC2617).* Cependant, l'implantation de référence utilise seulement le schéma d'authentification basique dans le cas particulier de l'installation d'un MIDlet. Toute autre réponse HTTP 401 ou 407 ne sera pas reconnue dans le but de renvoyer une requête avec les informations fournies.

4.4.2 Vulnérabilités dans KVM

Vulnérabilité par dépassement de tampon

Un dépassement de tampon est souvent la conséquence d'un manque de vérification de limites par le programmeur lors d'une écriture de données dans un tampon de taille fixe. Il en résulte que des données du tampon remplacent des données en mémoire. Dans le scénario classique, le tampon est situé dans la pile du programme, et la donnée remplacée est la valeur de l'adresse de retour de la *stack frame* courante. Cette adresse de retour est changée pour pointer de nouveau dans le tampon, et lorsque l'on revient de la fonction dans laquelle le dépassement se produit, le programme saute (jump) vers l'adresse de retour boguée, et commence à exécuter le code contenu dans le tampon. Le contenu du code est bien sûr minutieusement choisi par l'attaquant, celui-ci s'exécutant avec les mêmes privilèges que le programme d'origine.

En inspectant le code source de la machine virtuelle (KVM), une vulnérabilité par dépassement de mémoire peut être identifiée. Le code vulnérable, situé dans le fichier native.c, est affiché ci dessous.

```
void invokeNativeFunction(METHOD thisMethod) {
    // ...
    NativeFunctionPtr native = thisMethod->u.native.code;
    if (native == NULL) {
        // Native function not found; throw error
        // ...
        sprintf(str_buffer, "Native method %s::%s not found", className,
            methodName(thisMethod));
        // ...
        fprintf(stderr, "ALERT: %s\n", str_buffer);
    }
}
```

Ce code lance une exception si une méthode native est déclarée dans un fichier *class* sans en donner l'implantation correspondante. Ce code ne vérifie pas la taille du message qui sera stocké dans *str_ buffer*. En sachant que *str_ buffer* est une variable globale déclarée comme une chaîne de caractères dans le fichier main.c par

```
char str_buffer[512]; /* shared string buffer */
```

Il est clair que le code aura un comportement non prévu si la taille du string stocké dans *str_ buffer* excède 512 octets. Une partie du string sauvé

dans *str_ buffer* sera le nom de la méthode native invoquée. En sachant qu'aucune restriction n'est imposée par la machine virtuelle sur la taille du nom des méthodes, une vulnérabilité existe clairement ici.

Il est néanmoins important de se rendre compte que cette vulnérabilité n'est pas en elle-même très dangereuse pour le moment, et provoquera seulement un crash de la machine virtuelle (KVM). En effet, dans MIDP 2.0, un MIDlet n'est pas autorisé à déclarer ni implanter sa propre fonction native. Néanmoins, si de futures versions de MIDP implantent cette fonctionnalité, la vulnérabilité discutée dans cette section pourrait être exploitée par un MIDlet malveillant.

4.4.3 Vulnérabilités dans le cycle de vie d'un MIDlet

Comportement inapproprié d'un MIDlet

Un MIDlet sur un appareil mobile peut être dans un des états illustrés à la figure 4.4. C'est le rôle du développeur d'implanter ces différentes méthodes. Durant son cycle de vie, un MIDlet est autorisé à exécuter du code qui changera son état et en notifie le système. De plus, le MIDlet est responsable de définir les commandes qui permettront à l'utilisateur de terminer le MIDlet. L'utilisateur devra donc toujours faire l'hypothèse que le MIDlet se comporte normalement, ce qui peut mener à des abus qui ne sont pas nécessairement relatifs à la sécurité, mais qui peuvent causer de forts désagréments. Ce sont principalement les MIDlet ne définissant pas de bouton pour terminer l'application (bouton *exit*) à l'écran ou ceux ne se comportant pas normalement durant les phases de leur cycle de vie.

Un autre type de MIDlet ne se comportant pas normalement est celui qui définit un bouton de sortie à l'écran qui ne déclenche aucune action. Dans ce cas, c'est au système de l'appareil d'implémenter une fonction qui terminera le MIDlet, ou qui terminera la machine virtuelle.

Un exemple de MIDlet ayant ce comportement est décrit ci-dessous.

```
public class lifecycle extends javax.microedition.midlet.MIDlet implements
    CommandListener {
    private Command exitCommand;
    private TextBox tb;
    public lifecycle (){
        exitCommand = new Command("Exit",Command.EXIT, 1);
        tb = new TextBox("Hello Midlet", "hello world", 15, 0);
        Display.getDisplay(this).setCurrent(tb);
```

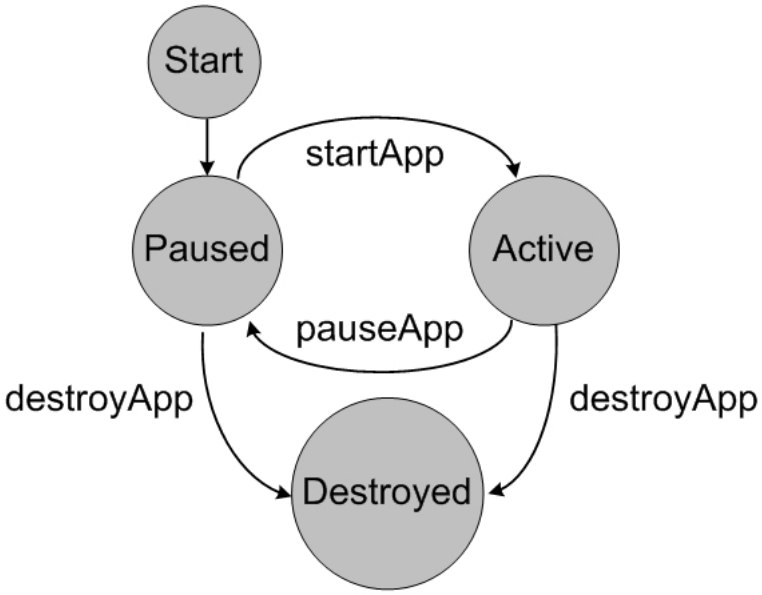

FIG. 4.4 – Cycle de vie d'un MIDlet.

```
        tb.addCommand(exitCommand);
        tb.setCommandListener(this);
    }
    public void startApp() { }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable d){
        if (c == exitCommand){
            destroyApp(false);
            notifyPaused();
        }
    }
}
```

Ici le MIDlet appelle la fonction destroyApp() pour se terminer (lorsque l'utilisateur presse le bouton exit). Cependant, le MIDlet informe le système qu'il est dans l'état de pause en appelant la fonction *notifyPaused()*. Il en résulte un gel (freeze) de l'application qui doit être terminée de force.

Différences d'exceptions dans MIDP 1.0 et MIDP 2.0

Certaines méthodes qui sont implémentées dans MIDP 2.0 lèvent des types d'exceptions différentes que celles implémentées dans MIDP 1.0. Par conséquent, un MIDlet écrit pour MIDP 1.0 et appelant ce type de méthode ne pourrait pas être capable de capturer une exception s'il était exécuté sur MIDP 2.0. Le code suivant illustre ce propos.

```
public class Exceptions_test extends javax.microedition.midlet.MIDlet
    implements CommandListener {
    private Command exitCommand;
    private TextBox tb;
    public Exceptions_test(){
        exitCommand = new Command("Exit",Command.EXIT, 1);
        tb = new TextBox("Hello Midlet", "hello world", 15, 0);
        Display.getDisplay(this).setCurrent(tb);
        tb.addCommand(exitCommand);
        tb.setCommandListener(this);
        try{
            RecordStore rs = RecordStore.openRecordStore("Sting to
                generate exception", true);
        }
        catch (RecordStoreException e){ }
    }
    public void startApp() { }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable d){
        if (c == exitCommand){
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

Ce MIDlet appelle la méthode *openRecordStore*. Dans MIDP 1.0 ce dernier ne lance que des exceptions de la classe *RecordStoreException* ou d'une classe qui hérite de celle-ci. Mais dans MIDP 2.0, lorsque le nom du *RecordStore* n'est pas valide, cela provoque une exception de la classe *IllegalArgumentException*, qui n'hérite pas de la classe *RecordStoreException*. Dès lors, ce code s'exécute parfaitement sur un MIDP 1.0, mais sur un MIDP 2.0, il provoque une exception *IllegalArgumentException*, alors que le MIDlet s'attend à récupérer un *RecordStoreException*. L'exception ne sera donc pas récupérée, et

causera le crash de l'application.

4.4.4 Vulnérabilité du système de stockage

L'unité de stockage dans Java ME CDLC est le *record store*. Chaque MIDlet peut avoir un ou plusieurs enregistrements (*record store*). Ceux-ci sont stockés dans l'appareil de manière permanente. Les *record store* sont identifiés par un chemin d'accès unique. La structure actuelle des *record store* sur l'appareil consiste en un en-tête (header) et un corps (body). L'en-tête contient des informations à propos du *record store* alors que le corps consiste en une suite de plusieurs tableaux contenant les données appelé *record*. Enfin, la partie de la plateforme Java responsable de manipuler ces enregistrements est appelé Record Management System (RMS). La figure 4.5 illustre cette structure.

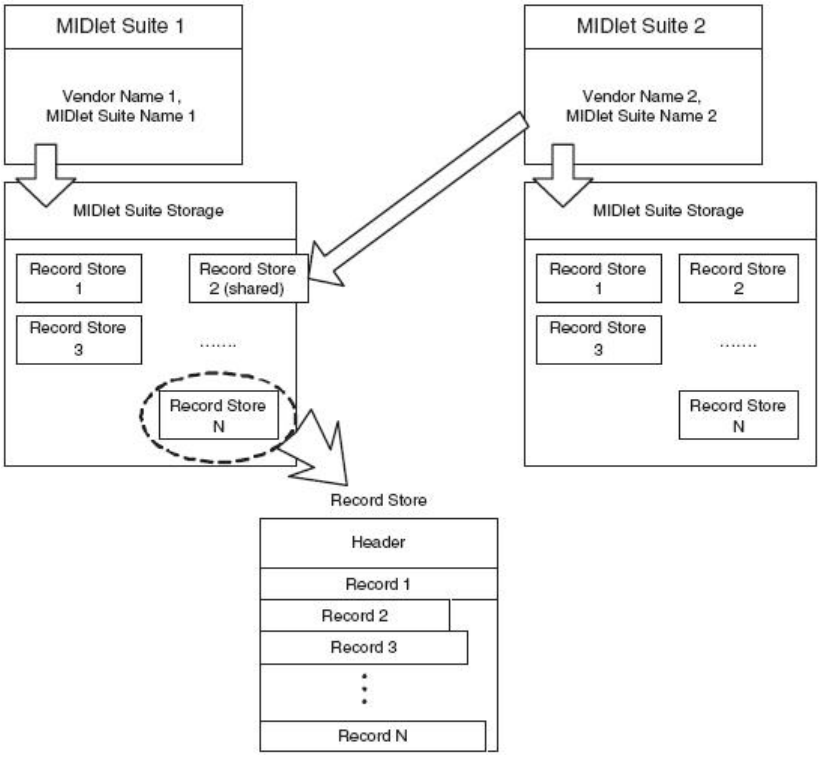

FIG. 4.5 – Structure du *record store*.

Données non protégées

Les données situées dans le *record store* ne sont pas protégées contre les attaques malveillantes. Il n'y a aucune mention dans la spécification de protéger les données sensibles des utilisateurs, comme les mots de passe. Les données peuvent être vulnérables à n'importe quelle attaque en dehors du RMS, comme par exemple lors d'un transfert de données vers un PC dans le cadre d'une sauvegarde. De plus, l'entièreté du système de stockage dans MIDP peut être accessible depuis le système de fichiers présent sur l'appareil, pouvant aussi compromettre des données sensibles.

Vulnérabilité dans la gestion de l'espace libre disponible

Lorsqu'un MIDlet souhaite stocker des informations permanentes dans le système de stockage, il peut y créer un nouvel enregistrement. Comme le système de stockage permanent est partagé par tous les MIDlets installés sur l'appareil, des restrictions doivent être prises sur la quantité de stockage attribuée à chacun. Cependant, il n'existe aucune restriction sur la taille de stockage allouée à un MIDlet dans les spécifications MIDP. Dès lors, il n'est pas possible d'empêcher un MIDlet de demander l'allocation de tout l'espace disponible, empêchant ainsi les autres MIDlets de pouvoir écrire des données permanentes, ce qui peut parfois être vital pour certaines applications. Un exemple de code demandant l'allocation de tout l'espace disponible est présenté ci-dessous.

```
// StartApp method //
public void startApp() {
    boolean enoughSpace=true;
    int  AvailableSize=0;
    RecordStore MainRS;
    try{
        MainRS = RecordStore.openRecordStore("attack"+NBcreatedRS, true
            );
        NBcreatedRS++;
    }
    catch (Exception e){
        enoughSpace=false;
    }
    while(enoughSpace){
        try{
            MainRS = RecordStore.openRecordStore("attack"+NBcreatedRS,
                true);
```

```
                NBcreatedRS++;
            }
            catch (Exception e){
                enoughSpace=false;
            }
        }
}
```

Ainsi, d'après les spécifications MIDP courantes, quelqu'un pourrait écrire une méthode prenant tout l'espace persistant disponible sur l'appareil, l'encapsuler dans un MIDlet quelconque (un jeu par exemple) et par là empêcher les autres MIDlets présents sur l'appareil de pouvoir demander de l'espace de stockage permanent. Il n'est bien sûr pas acceptable qu'un MIDlet puisse acquérir tout l'espace persistant pour lui, surtout quand on connaît les problèmes de ressources limitées de certaines appareils. Deux solutions pourraient être envisagées pour contrer ce problème :

- limiter la taille de la mémoire persistante que peut acquérir un MIDlet.
- rendre possible pour l'utilisateur de visualiser le montant d'espace mémoire pris par chaque MIDlet installé sur le système ; cette information pouvant aider l'utilisateur à détecter ce genre de MIDlet malveillant, et le cas échéant le supprimer.

API internes non protégées

Les API MIDP sont conçues pour fournir toutes les fonctionnalités nécessaires au développeur, et devraient être les seules API utilisables directement. Toutefois, ces API hauts niveaux ont été conçues pour rendre la programmation plus facile, et elles nécessitent l'aide d'API bas niveaux ainsi que de méthodes natives pour utiliser le matériel présent sur l'appareil. Les API bas niveaux sont plus proches du matériel de l'appareil, et par conséquent sont plus difficiles à programmer, mais possèdent plus de privilèges et moins de restrictions pour agir sur les pilotes du matériel de l'appareil. Cela ne devrait pas affecter la sécurité du système, l'accès à ces API devrait être restreint aux API de hauts niveaux. En d'autres termes, les développeurs ne devraient pas avoir accès à ces API de bas niveaux. Malheureusement, dans l'implantation de référence de SUN, ce n'est pas le cas.

Dans le système de stockage MIDP, une API de haut niveau est par exemple la classe *RecordStore*. Elle fournit les fonctionnalités dont a besoin le programmeur pour manipuler des enregistrements, comme l'ouverture, fermeture, suppression de fichier, ... Cette classe vérifie aussi les droits d'accès avant d'effectuer n'importe quelle action, dans le but évident de protéger la

sécurité et l'intégrité des données. Par exemple, aucun MIDlet n'est autorisé à supprimer les enregistrements d'un autre MIDlet. Il existe cependant une classe bas niveau, le *RecordStoreFile*. Cette classe est beaucoup plus proche des pilotes du matériel de l'appareil, appelant des méthodes natives et fournissant des services à la classe *RecordStore*. Cette classe ne devrait pas être directement disponible aux développeurs, car elle possède plus de privilèges et passe outre les vérifications de sécurité. Malheureusement cette classe est utilisable directement par le programmeur dans l'implantation de référence de SUN, ce qui peut compromettre la sécurité des données. Ainsi il est par exemple possible d'exploiter ceci pour développer un MIDlet capable de supprimer les enregistrements appartenant à un autre MIDlet !

Le code fourni en annexe exécute cette opération. Le premier MIDlet (listing A.1) crée un enregistrement, et le second (listing A.2) est capable de le supprimer avec succès.

4.4.5 Vulnérabilités réseaux

Vulnérabilité SSL dans MIDP

Dans le but d'établir une connexion sécurisée avec des sites distants (HTTPS), MIDP utilise le protocole SSLv3.0. L'implantation est basée sur KSSL de Sun. L'implantation MIDP de SSL (Secure Socket Layer) supporte les fonctionnalités suivantes :

- Une négociation SSL abrégée.
- Supporte seulement l'authentification serveur (pas d'authentification client).
- Le chiffrement RSA (Rivest Shamir Adleman) pour l'échange de clés.
- Le chiffrement RC4 Rivest Cipher 4.
- L'algorithme de hachage MD5 (Message Digest 5).
- L'algorithme de hachage SHA (Secure Hash Algorithm).
- Les certificats RSA X.509.

Durant la négociation SSL, le protocole doit générer des valeurs aléatoires utilisées pour créer le code secret principal (master secret). Le code secret principal est ensuite utilisé pour générer un ensemble de clés symétriques pour le chiffrement. Par conséquent, générer des valeurs aléatoires qui sont imprévisibles est un aspect important de la sécurité de SSL.

MIDP 2.0 ne fournit pas de génération de nombre pseudo-aléatoire (Pseudo Random Number Generator) robuste, similaire par exemple à la fonction *Se-*

cureRandom dans J2SE. La méthode pour générer des données aléatoires dans MIDP est *PRand.generateData* ; elle est présentée ci-dessous.

```
// PRand Class
public void generateData(byte abyte0[], short word0,short word1){
    synchronized(md){
        int i = 0;
        do{
            if(bytesAvailable == 0)
            {
                md.doFinal(seed, 0, seed.length,randomBytes, 0);
                updateSeed();
                bytesAvailable = randomBytes.length;
            }
            while(bytesAvailable > 0)
            {
                if(i == word1)
                { return; }
                abyte0[word0 + i] = randomBytes[--bytesAvailable];
                i++;
            }
        } while(true);
    }
}
```

La méthode applique un algorithme de hachage (MD5) sur le *seed* et met le résultat dans le tableau *randomBytes*. Ensuite, *randomBytes* est utilisé pour générer progressivement le tableau résultat (abyte0) qui sera renvoyé par la méthode *generateData*.

Plus précisément, la méthode *doFinal* applique l'algorithme MD5 sur le *seed* pour générer *randomBytes*. Comme MD5 est déterministe, s'il est appliqué sur le même *seed*, il génèrera la même valeur aléatoire. Par conséquent, le défi est de changer le *seed* d'une manière non prévisible.

La méthode utilisée dans MIDP pour mettre à jour le *seed* est appelée *updateSeed* et est fournie ci-dessous :

```
//PRand Class
public void updateSeed()
{
    long l = System.currentTimeMillis();
    byte abyte0[] = new byte[8];
    for(int i = 0; i < 8; i++)
    {
```

```
            abyte0[i] = (byte)(int)(l & 255L);
            l >>>= 8;
        }
        md.update(seed, 0, seed.length);
        md.doFinal(abyte0, 0, abyte0.length, seed, 0);
    }
```

Il est facile de remarquer que la mise à jour du *seed* dépend seulement du temps système (*System.currentTimeMillis*). Donc, dans le but d'obtenir la valeur aléatoire générée par le client, tout ce que doit faire l'attaquant est de deviner le temps système précis (en millisecondes) où la génération de la valeur aléatoire a été effectuée. Pour ce faire, des sniffers ethernet peuvent être utilisés. Ces outils (par exemple tcpdump) enregistrent de manière précise l'heure de chaque paquet. Cela permet à l'attaquant de deviner un très petit intervalle de temps correct.

Un scénario d'attaque basique peut être mené lorsque le client aura besoin d'envoyer un défi (challenge) au serveur. Le défi est certainement un *nonce* (une valeur aléatoire fraîche utilisée qu'une seule fois) chiffré avec la clé publique du serveur. Le *nonce* est un nombre aléatoire généré par l'algorithme présenté ci-dessus. Normalement, seul le serveur sera capable de déchiffrer le défi et de renvoyer la valeur correcte du *nonce*. Cependant, en utilisant la technique présentée pour générer le nombre aléatoire, l'attaquant peut répondre au défi sans connaître la clé publique du serveur. A cette fin, l'attaquant peut déterminer un intervalle de temps proche de la valeur correcte du *nonce*. Ensuite, il chiffre chaque valeur du *nonce* possible avec la clé publique du serveur. Enfin en comparant chaque valeur générée avec le défi reçu du client, il révélera la valeur correcte du nonce.

Dans l'implantation MIDP de SSL, la vulnérabilité dans l'algorithme de génération de nombre aléatoire peut être exploitée pour effectuer une attaque plus sophistiquée. Durant la négociation SSL, le client et le serveur se mettent d'accord sur un code secret principal à partir duquel toutes les clés symétriques seront générées. Le code secret principal est alors construit à partir de trois valeurs, à savoir *Client_Random_Value*, *Server_Random_Value* et le code secret préliminaire (premaster secret).

Client_Random_Value est une valeur aléatoire générée par le client et envoyée dans le message *ClientHello*. Comme *ClientHello* est envoyé en clair, récupérer cette valeur n'est pas difficile. *Server_Random_Value* est aussi générée par le serveur et envoyée en clair dans le message *ServerHello*.

Enfin, le code secret préliminaire (premaster secret) est généré par le client en utilisant le code suivant :

```
// Handshake Class
private void sendKeyExchange() {
    // ...
    preMaster = new byte[48];
    rnd.generateData(preMaster, (short)0, (short)48);
    preMaster[0] = (byte)(ver >>> 4);
    preMaster[1] = (byte)(ver & 0xf);
    // ...
}
```

Il est dès lors facile de remarquer que le code secret préliminaire est généré en utilisant l'algorithme de génération de nombre pseudo-aléatoire *generateData* discuté ci-avant et ensuite en changeant le premier et deuxième octet par des constantes. Cependant, à la différence de *Client_Random_Value* et *Server_Random_Value*, le code secret préliminaire est chiffré avec la clé publique du serveur avant son envoi au serveur.

Appliquer le scénario d'attaque basique discuté précédemment permet à un attaquant de trouver la valeur du code secret préliminaire sans connaître la clé publique du serveur. Par conséquent, l'attaquant possède maintenant toutes les valeurs nécessaires pour générer la code secret principal (master secret). A partir de là, il peut générer toutes les clés secrètes utilisées dans la communication entre le client et le serveur, lui permettant de mener à bien plusieurs attaques durant la session de communication, comme l'écoute, l'attaque par l'homme au milieu (Man-In-The-Middle) ...

4.4.6 Vulnérabilités dans le système de threading

Alors que le *multi-threading* est supporté, aucune mesure n'est prise pour synchroniser l'accès au système de stockage. Lorsque deux ou plusieurs *threads* essaient de lire ou écrire des données dans le système de stockage, l'intégrité des données ne peut être garantie. La synchronisation est laissée à la responsabilité du programmeur. Un MIDlet malveillant pourrait utiliser ce fait pour corrompre les données appartenant à un autre MIDlet (dans le cas de données partagées). De plus, l'intégrité des données stockées par un MIDlet dans son propre espace de stockage peut être compromise dans le cas où plusieurs *threads* essaient de lire ou d'écrire les mêmes données.

4.5 Conclusion

Avec l'augmentation et la popularité croissante des systèmes mobiles et sans fil, il y a une prolifération d'appareils connectés à Internet. Dans ce contexte, Java se singularise comme environnement standard d'exécution par sa sécurité, sa portabilité, sa mobilité et ses fonctionnalités de support réseau. La plateforme de choix dans cette configuration est Java ME CDLC. Elle est utilisée pour fournir nombre de services et applications : Services Web, jeux, messageries, commerce mobile, ...

Un autre facteur qui a amplifié l'adoption massive de Java ME est le vaste éventail de solutions Java déjà disponibles sur le marché. Tous ces facteurs ont fait de Java en général et Java ME en particulier une solution idéale pour le développement de logiciel dans le domaine des appareils mobiles.

Malgré son succès commercial et industriel, Java ME CLDC reste une plateforme très récente et sa sécurité doit continuer à être étudiée et évaluée. Dans ce chapitre, je n'ai pu parcourir tous les mécanismes de sécurité présents, certains ont simplement été cités (et d'autres moins en relation avec ce mémoire omis) et il faudrait un ouvrage entier pour les décrire tous. Néanmoins ce chapitre donne un bon aperçu des mécanismes mis en œuvre et présente les différentes attaques possible sur la version actuelle de Java ME CLDC.

Enfin, la prochaine version de MIDP (3.0) continuera à mettre l'accent sur la sécurité en ajoutant (encore) de nouvelles fonctionnalités de sécurité (le RMS sécurisé, le support de l'IPv6, ...)

Chapitre 5

Exploit dans Windows CE utilisant une architecture ARM

Windows CE est un des systèmes d'exploitation pour PDA, téléphones mobiles et smartphones les plus populaires. Alors qu'au niveau API, la plupart des appels systèmes et interfaces sont similaires aux versions standards de Windows, l'implantation interne diffère beaucoup étant donné les contraintes de support de différents types de processeurs et d'architectures.

Ce chapitre a pour but de présenter la création d'un exploit et le développement d'un *shellcode* pour Windows CE tournant sur une architecture ARM. Nous passerons en revue brièvement l'architecture ARM ainsi que l'architecture mémoire de Windows CE, et verrons les difficultés et les connaissances dont il faut tenir compte pour mener à bien cette réalisation.

Un *shellcode* est une chaîne de caractère qui représente un code binaire exécutable, destiné à être injecté dans une vulnérabilité (la plupart du temps dans une fonction de chaînes de caractères ne vérifiant pas un dépassement) et dont le but est souvent de pouvoir lancer un *shell* (une invite de commande).

Enfin ce chapitre est basé sur l'article original de [63]. Les illustrations de code donnés en exemple dans ce chapitre proviennent de [64].

5.1 ARM

Les processeurs ARM peuvent fonctionner soit en mode petit-boutiste (little-endian) soit en grand-boutiste (big-endian). Les processeurs récents fonctionnent et disposent d'un jeu d'instructions 32 bits, mais certaines versions (ARM7 et ARM9) disposent d'un second jeu d'instructions appelés

Registre	Alias	Description
R0		Registre général (Argument 0 et valeur de retour)
R1		Registre général (Argument 1 et seconde moitié de la valeur de retour)
R2,R3		Registres généraux (Argument 2 et 3)
R4-R10		Registres généraux
R11	FP	Frame Pointer (registre général)
R12		Registre général
R13	SP	Pointeur de pile (Stack Pointer)
R14	LR	Registre de lien (adresse de retour d'un appel de fonction)
R15	PC	Compteur ordinal (Program Counter)
	PSR	Registre de statuts (4 bits - Program Status Register)

TAB. 5.1 – Les registres des processeurs ARM.

Thumb permettant le codage d'instructions sur 16 bits et ainsi réaliser un gain de mémoire important notamment pour les applications embarquées.

Windows CE requiert que le processeur fonctionne en mode 32 bit petit-boutiste.

Les processeurs ARM pouvant faire tourner Windows CE possèdent 16 registres 32 bits (voir table 5.1). Le registre de statuts (program status register, PSR) comporte 4 bits, et contient les bits *négatif*, *zéro*, de *report* (carry) et de *dépassement* (overflow) (NZCO).

Ces processeurs possèdent trois dispositifs qui peuvent affecter le développement d'un *shellcode*, à savoir : le comportement des registres PC (Program Count - compteur ordinal) et SP (Stack Pointer - pointeur de pile), l'exécution conditionnelle d'instructions et la taille fixe des instructions.

Registres PC et SP

Le compteur ordinal et le pointeur de pile sont stockés dans 2 registres. Ils peuvent donc être facilement lus et modifiés. La possibilité de lire la valeur du compteur ordinal est utile, car le shellcode a besoin de connaître sa propre position en mémoire. Un exemple donné en annexe (Figure B.1) montre comment un programme peut déterminer l'adresse de la prochaine instruction à exécuter. Changer le compteur ordinal peut être intéressant pour éviter des instructions de branchement ou de saut (qui peuvent éventuellement contenir des octets NULL)

Exécution conditionnelle

L'exécution conditionnelle peut être utilisée pour réduire la taille du shellcode, car les instructions de comparaison peuvent souvent être évitées par

l'utilisation de l'exécution conditionnelle. Cette exécution peut aussi servir à éviter les octets NULL, ce qui est important dans la phase initiale de l'exécution du shellcode.

L'exécution conditionnelle permet d'éviter l'exécution de certaines instructions si une condition n'est pas remplie. Ces instructions doivent être manuellement écrites par le programmeur, et sont une alternative aux processeurs supportant la prédiction de branchement.

Un exemple est donné en annexe (Figure B.2).

Taille fixe des instructions

Les processeurs ARM utilisent des instructions de taille fixe qui peuvent poser deux problèmes lors du développement du shellcode. Tout d'abord, comme les instructions sont de taille fixe, il n'y a pas d'instruction *No Operation* (NOP, instruction signalant le processeur qu'il ne doit rien faire). Cette instruction est normalement utilisée pour l'alignement du code (pipe-line), dont n'ont pas besoin les processeurs ARM.

Or un exploit utilise l'instruction NOP pour créer un *NOP-Sled*, une zone située avant le shellcode et contenant un nombre significatif d'instructions NOP. Cette zone permet d'ajouter de la souplesse à l'exploit, car la localisation exacte de l'exploit peut quelque peu varier à chaque exécution. En précisant l'adresse de retour au milieu de la zone *NOP-Sled*, de petites variations de position en mémoire de l'exploit peuvent être tolérées. L'instruction NOP peut être remplacée en utilisant une instruction n'ayant aucun effet (ne changeant aucun registre). La figure B.3 donne deux exemples.

Un deuxième problème est que les valeurs immédiates sont limitées à 8 bits dans le jeu d'instructions ARM (12 bits maximum en incluant un glissement (shift) de 4 bits possible). Une solution pour utiliser des valeurs immédiates de 32 bits est de créer une section de données dans le shellcode contenant les valeurs de 32 bits. Ces valeurs peuvent alors être chargées en utilisant l'instruction LDR (load).

5.2 Windows CE

Windows CE a déjà été décrit succinctement précédemment. Seule la partie intéressante pour le développement de l'exploit sera discutée ici. La version analysée ici sera Windows CE 4.2 (Windows CE .NET)

Windows CE supporte un maximum de 32 processus simultanés, et un nombre virtuellement illimité de *threads* pour chaque processus (cependant limité par la quantité de mémoire disponible). L'API de Windows CE est principalement basée sur une bibliothèque de liens dynamiques (dynamic link libraries - DLL) au lieu d'appels systèmes. Ces appels systèmes existent, mais sont rarement utilisés par les applications habituelles.

5.2.1 Architecture mémoire

Windows CE possède un espace mémoire virtuel adressable de 32 bits (4GB) divisé en deux parties. La partie haute de 2 GB est l'espace pour le noyau (kernel) et la partie basse, l'espace utilisateur (2 GB).

L'espace utilisateur est alors divisé en 64 *slots* de 32 MB chacun, où les slots 0 à 32 sont utilisés pour les processus en cours d'exécution. La figure 5.1 illustre l'agencement de l'espace mémoire.

Les slots 0 et 1 sont réservés. Le slot 0 est un slot virtuel, lié au processus actif du système situé dans un autre slot. Le slot 1 contient la bibliothèque de liens dynamiques du système. Donc les slots 2 à 32 contiennent les processus endormis. Ces slots sont assignés de bas en haut, et un nouveau processus sera toujours placé dans le plus petit slot disponible.

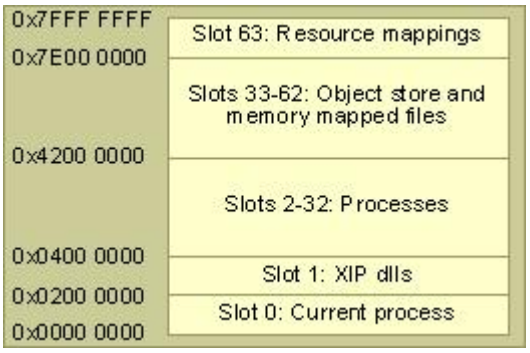

FIG. 5.1 – Architecture mémoire de l'espace utilisateur de Windows CE.

5.2.2 Bibliothèque de liens dynamiques (DLL)

Windows CE utilise une technique appelée *eXecute In Place* (XiP) pour toutes les DLL du système (celles qui sont à l'origine installées dans l'appa-

reil). Les XiP-DLL sont stockées dans la ROM et sont associées (mapped) en mémoire avec le slot 1 avec seulement les permissions de lecture et d'exécution. De plus, XiP implique que chaque DLL soit toujours associée à la même adresse virtuelle dans un appareil.

Les DLL fournies par un programme tiers sont chargées en mémoire dans l'espace du processus les utilisant à partir de l'adresse 0x1FFFFFF (pour le slot 0). Si plusieurs DLL tierces sont utilisées, leur ordre en mémoire reste constant à travers plusieurs exécutions de l'application correspondante.

5.2.3 Appels de fonctions

Les appels de fonctions et de bibliothèques fonctionnent de façon similaire : tout d'abord, les arguments sont placés dans les registres appropriés (R0 à R3), et après le compteur ordinal (PC) est modifié pour continuer l'exécution à l'adresse de la fonction. Si une fonction requiert plus de quatre arguments, les arguments 5 à N sont placés sur la pile.

Les appels de bibliothèques présentent une étape supplémentaire, car celles-ci sont associées trop loin en mémoire pour être directement adressables avec une instruction de branchement. L'étape intermédiaire consiste alors à se brancher sur une table de bibliothèques qui charge directement l'adresse de la fonction dans le compteur ordinal (PC).

Une fonction est appelée par une instruction de branchement (BL), qui va stoker l'adresse de retour dans le registre de lien (LR) et ensuite modifier le compteur ordinal (PC) pour continuer l'exécution à l'adresse spécifiée. Le prologue consiste donc à sauver le pointeur de pile (SP) et le registre de lien (LR) sur la pile avant d'allouer l'espace pour les variables locales. Au retour de fonction, l'épilogue consiste à détruire les variables locales en modifiant le pointeur de pile (SP), et à restaurer l'ancien pointeur de pile et compteur ordinal. Le tableau 5.2 illustre le prologue et l'épilogue.

L'adresse de retour (sauvée dans LR) est alors la première valeur contenue sur la pile de la fonction courante.

5.3 Développement de l'exploit/shellcode

Les attaques par dépassement de tampon (buffer overflow) sur Windows CE fonctionnent de façon assez semblable à celles sur les architectures x86

@prologue	
MOV R12, SP	R12 = SP
STMFD SP !, R12, LR	sauve R12 et LR (bouge la pile de 8 octets)
SUB SP, SP, #200	200 octets pour les variables locales
...	
@épilogue ADD SP, SP, #200	destruction des variables locales
LDMFD SP, SP, PC	restauration de SP et PC

Tab. 5.2 – Prologue et épilogue des appels de fonctions.

classiques. L'adresse de retour sur la pile est écrasée pour rediriger le flot du programme vers le shellcode sur la pile. Le schéma classique sur la pile est alors **[zone NOP-Sled] [Shellcode] [Adresse de retour]**. Néanmoins quelques complications existent et seront discutées ci-après.

La zone NOP-Sled peut être implémentée en utilisant des instructions n'ayant aucun effet sur le processeur, comme décrit sur la figure B.3. L'adresse de retour devra pointer quelque part dans la zone NOP-Sled.

5.3.1 Shellcode

Créer un shellcode pour Windows CE n'est pas si aisé, dans le sens où aucune invite de commande n'est disponible. Par conséquent, le shellcode doit contenir tout le code nécessaire pour implémenter les fonctionnalités désirées. En outre, le shellcode pour Windows CE doit utiliser des appels aux bibliothèques à la place d'appels systèmes pour accéder aux fonctionnalités du système d'exploitation, comme l'accès au système de fichier ou aux interfaces réseaux.

Or, utiliser des fonctions DLL dans le shellcode n'est pas simple, car cela implique de connaître l'adresse de la fonction de la bibliothèque. Deux solutions existent pour découvrir ces adresses : la méthode *statique* et la méthode *dynamique*.

Méthode statique

La méthode statique implique d'acquérir les adresses de fonctions au préalable, et de les placer ainsi dans le shellcode. Alors que le shellcode généré de cette manière est de petite taille, des inconvénients existent. En effet l'exploit ne pourra alors fonctionner que sur des appareils possédant les mêmes configurations de DLL et, par conséquent, l'exploit ne fonctionnera alors qu'en-

vers un type spécifique d'appareil. La table B.1 montre à titre informatif les adresses de fonctions de deux appareils mobiles.

Méthode dynamique

La méthode de recherche dynamique implique d'incorporer davantage de code pour pouvoir effectuer une recherche dans la liste des DLL du noyau. Cependant, un exploit qui utilise cette technique pourra fonctionner sur tous les appareils.

hex	assembleur	commentaire
@CODE		
0x18C09FE5	LDR R12, [PC, #24]	charge l'adresse de MessageBoxW dans R12
0x000020E0	EOR R0, R0, R0	Met R0 à 0
0x14108FE2	ADD R1, PC, #20	Met R1 à l'adresse de "Update..."
0x34208FE2	ADD R2, PC, #52	Met R2 à l'adresse de "You got..."
0x1230A0E3	MOV R3, #1	Met R3 à 1
0x0FE0A0E1	MOV LR, PC	sauve l'adresse de retour dans LR
0x0CF0A0E1	MOV PC, R12	exécute MessageBoxW
0x24F04FE2	SUB PC,PC, #36	JUMP à la première instruction (boucle)
@DATA		
0x0098F800		adresse de MessageBoxW
'U',0,'p',0,'d',0,'a',0,'t',0,'e',0,...		
'Y',0,'o',0,'u',0,' ',0,'g',0,'o',0,'t',0,...		

Tab. 5.3 – Shellcode affichant une boîte de dialogue.

La figure 5.3 montre un exemple de shellcode qui affiche une boîte de message. Ce shellcode utilise la méthode statique. Il est séparé en deux, CODE et DATA. La partie CODE contient le code exécutable, et la partie DATA contient l'adresse de la fonction de la bibliothèque MessageBoxW et le string utilisé pour le titre de la boîte ainsi que le texte du message.

La première instruction charge l'adresse de la fonction MessageBoxW dans R12. La seconde instruction met R0 à 0. La troisième instruction met R1 à l'adresse du premier string en ajoutant l'offset au compteur ordinal (PC). La quatrième instruction met R2 à l'adresse du deuxième string. La cinquième instruction met R3 à 1. La sixième instruction sauve l'adresse de retour dans LR, et la septième instruction met le compteur ordinal (PC) à l'adresse de la fonction MessageBoxW, et l'exécute. La huitième instruction est appelée après le retour de fonction, elle remet le compteur ordinal à l'adresse de la première instruction, créant ainsi une boucle. La figure 5.2 montre le résultat réel.

FIG. 5.2 – Résultat de l'exécution du shellcode.

5.3.2 Le problème de l'octet NULL

L'octet zéro ou NULL constitue un problème dans un shellcode car les fonctions de chaînes de caractères, souvent utilisées comme injecteurs de code, arrêteront de copier les octets en présence d'un octet NULL. Par conséquent, l'exploit a besoin d'être complètement débarassé de tout octet NULL.

Les exploits peuvent néanmoins être modifiés dans le but d'éviter des octets NULL, mais dans le cas de Windows CE ceci est plus complexe, car ils doivent faire face à des sources additionnelles d'octets NULL. En effet, sur l'architecture ARM, un octet NULL est ajouté par chaque instruction utilisant le registre R0 (qui est le premier argument d'un appel de fonction ou de bibliothèque). De plus, Windows CE utilise intensivement l'Unicode. Le codage Unicode utilise 16 bits pour représenter un caractère, créant ainsi un octet NULL pour chaque caractère ASCII (plus précisément ici c'est certainement le codage ucs-16 d'Unicode, mais je n'ai pas trouvé de confirmation).

Une solution pour le problème de l'octet NULL est d'utiliser une méthode de *codage par XOR*. Le code du shellcode est alors *XORer* avec une clé dans le but d'éliminer tous les octets NULL. Quand l'exploit est exécuté, une routine de décodage XOR le code avant de sauter en lui. Remarquons que le décodeur lui-même se doit d'être sans octet NULL.

Néanmoins cette solution soulève un problème : les processeurs ARM utilisent une architecture de type *Harvard*, c'est-à-dire une architecture dans laquelle le bus de données et le bus d'instructions sont séparés, chacun possédant sa propre antémémoire. Le problème avec un code auto-modifiant est que les opérations de *load/store* utilisées pour décoder le code principal ne modifient que la cache de données, la cache d'instructions contenant toujours la version non modifiée (non décodée) du shellcode. C'est la conséquence directe du fait que le shellcode codé suit directement le code du décodeur, et par conséquent, celui-ci est déjà chargé dans la cache d'instructions.

La solution à ce problème requiert alors de déplacer le shellcode décodé plus en avant dans la pile, dans une région qui n'a pas encore été chargée dans la cache d'instructions. Il en résulte que le shellcode décodé pourra alors être enfin chargé et exécuté.

5.3.3 Complication

Il existe encore deux problèmes additionnels dont il faut tenir compte.

Premièrement, sur l'architecture ARM, il est requis que le pointeur de pile (SP) soit plus petit ou égal au compteur ordinal (PC, SP <= PC) pour pouvoir exécuter n'importe quelle instruction.

Deuxièmement, un autre problème a été observé dans [64]. Il peut arriver qu'une partie des données de la pile deviennent corrompues (les données changent sans que l'on en connaisse la raison) au retour d'un appel de fonction, après avoir changé le pointeur de pile, mais avant de charger les valeurs précédentes de SP et PC qui se trouvent sur la pile. Cela n'a pas d'incidence sur l'exécution normale d'une application, mais cela peut entraîner la corruption de l'exploit, car le shellcode est injecté sur la pile.

5.4 Faisabilité de l'exploit

Attaquer Windows CE en pratique est plus difficile qu'attaquer un Windows sur une architecture x86 à cause de l'architecture mémoire de Windows CE, et en particulier à cause de la structure des slots de processus.

Chaque processus utilise un slot (situé dans une région bien précise de la mémoire globale) au lieu d'avoir son propre espace virtuel. Dans le but d'attaquer une application, le slot correspondant à cette application doit être connu, car l'adresse de retour injectée doit pointer exactement dans la

région mémoire de ce slot. Bien que le processus actif courant soit associé au slot 0, celui-ci ne peut être utilisé comme solution car l'adresse du slot 0 contient un octet NULL (il ne sera alors utilisé que si un octet NULL ne pose aucun problème pour l'exploit, rendant toutes les attaques par dépassement de tampon sur les fonctions de chaînes de caractères inutiles).

Il n'existe pas de moyens miracles pour connaître à l'avance le slot qu'occupera une application, autrement qu'en le devinant (on peut utiliser des outils de développement ou de débogage pour le connaître a posteriori). Néanmoins deviner le numéro de slot n'est pas infaisable, car la méthode d'allocation de slot est connue et un nouveau processus est toujours placé dans le slot le plus petit disponible. De plus, le nombre de slots total est petit (32) et bien souvent la moitié de ceux-ci sont déjà occupés par des processus en tâche de fond.

Les processus système utilisent une attribution de slots fixes sur chaque appareil et ne changent jamais. Ainsi ils peuvent être exploités facilement et constituent des cibles de choix, car le numéro de slot peut être déterminé en examinant à l'aide d'un outil de développement un appareil similaire à la cible.

Chapitre 6

Attaques par services croisés

Ce chapitre présente les attaques par services croisés (cross-services attacks), un nouveau type d'attaque qui cible spécifiquement les smartphones possédant de multiples interfaces de communications (généralement sans fil), et l'illustration d'un mécanisme de protection possible basé sur les travaux de [70] pour empêcher ce type d'attaque.

6.1 Introduction

Les appareils mobiles comme les PDA et les téléphones mobiles ont convergé en un nouveau type d'appareil. Ces nouveaux appareils, les smartphones, intègrent différentes technologies sans fil, comme le WLAN, le Bluetooth, et le GSM/GPRS. Malheureusement, l'intégration de ces différentes technologies est souvent effectuée en incluant simplement le matériel et logiciel nécessaires à leur fonctionnement dans l'appareil, sans tenir compte des différentes caractéristiques et des différents services liés à chaque technologie. Par conséquent, ces nouveaux appareils fortement intégrés peuvent être vulnérables face à un nouveau type d'attaque qui tire parti de l'interaction entre ces différents services.

Un exemple notable est l'interaction entre des services gratuits et des services qui requièrent un abonnement payant. Les téléphones cellulaires sont liés à un opérateur par un abonnement où l'utilisateur est facturé suivant le temps de communication ou suivant le volume de données transférées. Les PDA, d'un autre côté, supporte généralement un accès gratuit aux réseaux locaux IP à travers un accès avec ou sans fil.

Alors que les opérateurs de services cellulaires implémentent des pares-feu et autre formes de protection pour garantir la sécurité des utilisateurs, une protection moins stricte est généralement fournie lors de l'accès à des réseaux locaux (LAN). Dès lors, un appareil fortement intégré pourrait être compromis en exploitant la connexion au réseau local, permettant ainsi d'accéder à des services payants pouvant causer des dommages financiers à l'utilisateur.

Pour montrer la faisabilité de ces attaques sophistiquées contre les appareils qui intègrent des fonctionnalités de téléphone cellulaire et des fonctionnalités de PDA, nous partirons de l'exploitation d'une vulnérabilité par dépassement de tampon dans un service réseau, à travers une interface sans fil. Le code malveillant sera alors capable d'accéder aux fonctionnalités de téléphone cellulaire et d'effectuer des appels téléphoniques pour le compte de l'attaquant.

Même si les attaques par dépassement de tampon ne sont pas un nouveau concept, ce chapitre, basé essentiellement sur le travail de [70], présente une des premières descriptions détaillées de ce qu'une attaque par services croisés peut entraîner.

Les mécanismes de sécurité déployés sur les appareils mobiles ne permettent pas d'empêcher ce type d'attaque. Pour combler cette lacune, les auteurs de [70] ont décidés de créer un mécanisme qui compartimente l'accès aux ressources systèmes. Le but de ce mécanisme est d'éviter qu'un processus qui interagit avec un service réseau particulier puisse dépasser les limites de ce service et accéder aux ressources associés à d'autres services.

Ce mécanisme contrôle les appels systèmes exécutés par un processus et étiquette le code exécuté en se basant sur son accès aux interfaces réseaux. Lorsqu'une opération sensible est effectuée, les étiquettes des ressources invoquées sont comparées à un ensemble de règles. Ces règles permettent au développeur de spécifier de manière précise le contrôle d'accès aux services et aux données. Par exemple, il est possible de restreindre l'accès d'une application de carnet d'adresses qu'aux API destinés à l'appel téléphonique, et d'interdire l'accès aux API qui ne sont pas lié à l'application (les API de socket). L'étiquetage des ressources et des processus, ainsi que l'application des règles, sont effectuées par un dispositif de surveillance au niveau du noyau.

Pour que ce mécanisme soit général et facilement configurable, un langage de règles est défini qui autorise le développeur à spécifier quelles sont les actions autorisées par une certaine classe de programmes envers certains types de ressources.

6.2 Faisabilité de l'attaque

L'attaque documentée dans cette section montre comment il est possible de rentrer dans un smartphone par son interface LAN sans fil, et ensuite d'accéder à l'interface téléphonique de l'appareil pour composer un numéro. L'attaque sera dirigée vers un serveur ftp open-source (ftpsrv) sur un appareil mobile faisant tourner Windows Mobile Pocket PC.

6.2.1 Scénario de l'attaque

Un scénario possible serait une attaque par surfacturation envers un utilisateur abonné à un service, où l'appareil de la victime est utilisé pour effectuer des appels téléphoniques surtaxés (vers un numéro 0900 par exemple). D'autres attaques sont possibles, mais le fait qu'une attaque par surfacturation puisse générer des revenus pour l'attaquant (et des pertes pour la victime) suggère qu'elle sera sûrement répandue dans un futur proche.

Pour illustrer un exemple d'attaque, on peut imaginer un homme d'affaire voyageant dans différents lieux publics à la recherche d'un accès internet sans fil pour pouvoir récupérer ses emails. L'homme d'affaire trouve finalement une connection Internet sans fil dans un café et s'y branche. L'attaquant surveille justement le réseau sans fil du café et voit le nouvel appareil qui communique avec le point d'accès. Celui-ci scanne directement ce nouvel appareil et y découvre une vulnérabilité bien connue. Utilisant un exploit publié précédemment sur une liste de diffusion, l'attaquant arrive à accéder a l'appareil. L'exploit exécute alors un code qui compose un numéro 0900 possédé par l'attaquant, facturant un énorme montant à la victime.

6.2.2 Utilisation d'une vulnérabilité dans ftpsrv

Les auteurs de [70] ont choisi d'utiliser une vulnérabilité par dépassement de tampon car ceci constitue un type d'exposition courant sur toutes les plateformes.

Ils ont ensuite cherché quelles applications sur Windows CE pourraient posséder une telle vulnérabilité, en se concentrant sur des applications pouvant accepter des connexions entrantes. De nombreuses applications de ce type sont disponibles pour Windows CE. Ftpsrv fut finalement choisi, un serveur FTP open-source, car celui-ci contient une vulnérabilité par dépassement de tampon qui peut être utilisée pour réaliser l'attaque par services

croisés. Les détails de cette vulnérabilité sont données dans la section suivante.

6.2.3 Exploiter la vulnérabilité

La faiblesse utilisée pour l'attaque est une simple vulnérabilité *strcpy* situé dans la fonction ***void*** *Session::SendToClient(**int** mode, LPCSTR msg)* dans le fichier *ftpmain.cpp*. La fonction est appelée pour répondre aux commandes du client, qui dans certains cas renvoie les données fournies par ce client. L'attaque utilise la commande *USER* et le gestionnaire d'erreurs de commandes inconnues. Ces deux opérations utilisent la fonction *SendToClient*. L'invocation de *strcpy* dans la fonction *SendToClient* utilise un tampon fixe de 256 octets, qui autorise donc l'écrasement de l'adresse de retours dans le *stack frame* de la fonction.

A cause de corruptions aléatoires de l'ancien *stack frame* à la sortie de la fonction, il convient d'abord de placer le shellcode dans un endroit sûr. Pour cela on utilise le gestionnaire de commandes inconnues. Le gestionnaire sauvegarde la chaîne de caractères qui ne correspond à aucune commande dans la variable globale *m_szSjis* juste avant d'envoyer un message d'erreur au client. La modification du compteur ordinal est réalisée en utilisant la commande *USER*, qui écrase l'adresse de retour avec l'adresse de *m_szSjis*.

Une fois le shellcode exécuté, celui-ci compose un appel téléphonique. Ceci est accompli en deux étapes. Premièrement, la bibliothèque de téléphone est chargée (*mappée*) dans l'espace adressable de l'application en appelant la fonction *LoadLibraryW(TEXT("cellcore"))*. Deuxièmement, l'appel téléphonique est effectué en appelant la fonction *tapiRequestMakeCall*, qui compose le numéro donné en paramètre.

6.3 Empêcher l'attaque par l'étiquetage

L'exploit décrit dans la section précédente montre comment un attaquant peut abuser des ressources d'un appareil intégrant téléphone cellulaire et PDA.

Les solutions traditionnelles, comme des mécanismes de protection de pile, requièrent le support du compilateur et ne sont pas encore largement disponible pour les appareils utilisant Windows CE. Même si la version 5.0 de l'environnement de construction de Windows CE possède une option pour se protéger contre les attaques de pile (stack-smashing), cette fonctionnalité

n'est pas activée par défaut. De plus, des attaques par services croisés peuvent être menées sans effectuer de dépassement de tampon (en utilisant des erreurs de logiques situés dans des applications par exemple), et il convient donc de trouver une solution qui puisse empêcher directement cette attaque.

Les auteurs de [70] ont donc développé un mécanisme de sécurité basé sur l'étiquetage des processus et des ressources système. Dans ce qui suit, j'expliquerai les grandes lignes de ce mécanisme.

Celui-ci définit trois types d'objets, les processus p, les ressources r et les interfaces i. Les processus et ressources possèdent un ensemble associé d'étiquettes. Chaque étiquette représente le fait que soit directement, soit indirectement, le processus ou la ressource ait été en contact avec une interface réseau spécifique. On définit $L(i)$ l'étiquette associée avec l'interface i, et $LS(p)$ et $LS(r)$ l'ensemble des étiquettes associées respectivement avec un processus p et avec une ressource r.

Le mécanisme de sécurité inclut un composant de surveillance qui intercepte les appels systèmes potentiellement dangereux effectués par les processus. Ce sont les appels systèmes qui accèdent aux interfaces, qui accèdent ou exécutent les ressources, qui créent des ressources ainsi que de nouveaux processus. Quand un des ces appels systèmes est intercepté, les étiquettes du processus associé sont examinées en tenant compte d'un fichier global de règles qui spécifie quels types d'actions sont permises, suivant les étiquettes associés au processus. Le résultat de l'analyse peut établir que l'accès est refusé, que l'accès est autorisé ou que l'accès est autorisé, et, de plus, les étiquettes des ressources / processus impliqués dans l'opération sont modifiées. Dans ce qui suit, une description plus détaillée des opérations effectuées par le mécanisme d'étiquetage est présentée.

Accès aux interfaces

Lorsqu'un processus accède à une interface, les étiquettes du processus sont examinées pour déterminer si l'accès doit être autorisé. Si c'est le cas, le processus est marqué avec une étiquette représentant l'interface qui est accédée, c'est-à-dire $LS(p) = LS(p) \cup L(i)$, où p est le processus qui accède à l'interface i. Par exemple, si un processus accède à l'interface LAN sans fil en effectuant un appel système de socket, alors le processus est marqué avec l'étiquette qui spécifie l'interface LAN sans fil.

Accès aux ressources

Lorsqu'un processus demande l'accès à une ressource (par exemple pour ouvrir un fichier) les étiquettes associées avec le processus et la ressource sont examinées et tenant compte des règles existantes. Si l'accès est autorisé, alors l'ensemble des étiquettes du processus sont mises à jour avec l'étiquette de la ressource, c'est-à-dire $LS(p) = LS(p) \cup LS(r)$, où p et r sont respectivement le processus et la ressource impliqués.

Création de ressource et de processus

Lorsqu'un processus p crée une nouvelle ressource ou en modifie une existante, disons r, la ressource hérite de l'ensemble d'étiquettes du processus, tel que $LS(r) = LS(p)$. D'une manière similaire, lorsqu'un processus p crée un nouveau processus p', les étiquettes sont copiées vers le nouveau processus, ainsi $LS(p') = LS(p)$.

Le comportement du système d'étiquettes décrit ci-dessus autorise le mécanisme de sécurité à garder les traces de l'interface qui était impliquée et du processus ou de la ressource qui avait été affecté par les actions dont dépend la sécurité. Par exemple, si un processus lié à une certaines interfaces est compromis, les fichiers (et les processus) créés par le processus compromis seront marqués avec l'étiquette associée à l'interface. Ainsi, lorsque le processus compromis (ou le processus qui est créé par le processus compromis ou qui accède ou exécute une ressource créé par le processus compromis) essaie d'accéder à d'autres interfaces, il est possible d'identifier et de bloquer ces essais.

6.3.1 Spécifications des règles

Le mécanisme de sécurité utilise un fichier de règles pour déterminer s'il faut autoriser ou interdire un processus d'accéder à une ressource ou une interface. De plus, le fichier de règles peut être utilisé pour modifier le comportement d'étiquetage par défaut décrit ci-dessus.

Le contrôle d'accès est réalisé en spécifiant quelle(s) étiquette(s) un processus n'est pas autorisé à avoir lorsqu'il tente d'accéder à une interface ou une ressource. Par défaut, l'accès est autorisé à toutes les interfaces et toutes les ressources. Bien sûr, cette politique par défaut n'est pas très sécurisée, mais on anticipe ici le fait que les fournisseurs de services créeront des règles

compréhensives pour leurs utilisateurs, ou que leurs utilisateurs chevronnées adopteront des règles plus restrictives, si besoin.

Le fichier de règles consiste en une suite de règles, où chaque règle est composée de l'interface ou de la ressource cible, l'action à effectuer par le gestionnaire lorsque l'accès est demandé, et les étiquettes qui déclenchent l'action. Le langage de contrôle d'accès est définit comme suit :

$policy \Rightarrow rule*$

$rule \Rightarrow access\,(interface \mid ressource)\,action\,label*$

$action \Rightarrow deny \mid ask$

L'action *deny* interdit simplement l'accès, alors que l'action *ask* demande une confirmation à l'utilisateur à travers une boîte de dialogue. Par exemple une règle du type :

access i1 deny i2 i3

interdira l'accès à l'interface *i1* si le processus fut précédemment étiqueté avec les étiquettes associées aux interfaces *i2* ou *i3*.

Comme indiqué précédemment, le fichier de règles peut également être utilisé pour modifier le comportement d'étiquetage par défaut. Par défaut, chaque processus est étiqueté lorsqu'il accède à une interface (ou une autre ressource étiquetée) et lorsqu'il est créé par un processus déjà étiqueté. De plus, le langage de règles peut aussi être utilisé pour définir quelles applications sont exclues de ce comportement. Pour ce faire, trois types d'exceptions sont implémentées : l'exception *notlabel*, *notinherit* et *notpass*. Je ne détaillerai pas ici le fonctionnement précis des ces exceptions, le lecteur intéressé pourra lire l'article de référence pour de plus amples informations.

6.4 Implantation

Même si la preuve de faisabilité de l'exploit a été établie sur un système Windows CE, les auteurs ont implémenté leur prototype de système d'étiquetage sur la distribution Familiar Linux, car ils avaient besoin de modifier le noyau du système d'exploitation pour y ajouter quelques fonctionnalités. Le gestionnaire du système d'étiquetage tourne donc dans le noyau du système d'exploitation, à l'abri de quelconques modifications, à moins que le compte root soit compromis.

Les détails d'implantation ne seront pas donnés ici, le lecteur pourra avoir plus de détails en consultant l'article de référence.

6.5 Evaluation

En reprenant le type d'exploit cité au début du chapitre, mais cette fois-ci exécuté dans l'environnement Familiar Linux modifié avec la prise en compte du mécanisme d'étiquetage, nous verrons ci-après quelles en sont les conséquences.

Le processus qui fait office de serveur vulnérable est étiqueté lors de la création d'un socket (lorsque le processus invoque *socket(AF_INET, ...)*). Comme on ne peut réellement déterminer quelle interface sera utilisée comme connexion IP, les étiquettes associées au Wi-Fi et à l'Ethernet seront assignées.

Lorsque le code de l'exploit essaie d'accéder au port associé à l'interface GSM en utilisant l'appel système *open(2)*, le gestionnaire d'étiquetage est invoqué. Celui-ci compare alors les étiquettes du processus avec les règles associées dans le fichier de configuration. Le gestionnaire interdit l'accès, et l'appel système *open(2)* échoue.

Il convient néanmoins de noter que le dépassement de tampon a toujours lieu, et l'application va vraisemblablement *crasher*. Cependant, l'attaque par surfacturation ne peut être effectuée.

Comme remarqué auparavant, des protections d'intégrité de la pile et autres solutions orthogonales peuvent aider à empêcher un dépassement de tampon. Cependant, il y a d'autres types de vulnérabilités pour lesquelles ces techniques sont inefficaces. La solution du mécanisme d'étiquetage est générale, simple et efficace. Elle donne l'assurance que ces attaques n'ont qu'un impact limité, et empêche le franchissement des frontières d'une interface pour accéder à une autre interface.

Un cas spécial d'une application ayant besoin d'accéder à de multiples interfaces de classes différentes (une interface réseau IP et une interface cellulaire par exemple) pourrait poser problème pour ce système. Un exemple de ce type de situation serait une application téléphonique qui aurait besoin d'accéder à l'interface GSM et Bluetooth pour utiliser un casque sans fil dans le but d'une utilisation du GSM en kit mains libres. Un autre exemple serait la prochaine génération de téléphone hyper connectés, pouvant se connecter de préférence à un réseau Wi-Fi pour effectuer un appel téléphonique via la VoIP, et à un réseau GSM si il n'y a pas de connexion Wi-Fi disponible. Ce genre de situations peut être résolu en utilisant la règle d'exception *notlabel* pour ces application spécifiques. La règle interdira alors l'étiquetage de l'application lorsqu'elle accède à n'importe quelle interface, et dès lors elle sera capable de s'y connecter.

Enfin il faut également remarquer que l'utilisation de ce mécanisme d'étiquette implique une petite surcharge dans le système, notamment lorsqu'une application gère de nombreuses connections. Mais ceci constitue peut être le prix de la sécurité.

Chapitre 7

Autres menaces

Ce chapitre présente un aperçu des recherches conduites en matière d'appareils mobiles dont je n'ai pas parlé dans ce mémoire.

7.1 Sécurité des systèmes d'exploitation

La sécurité des systèmes d'exploitation des appareils mobiles a été analysée par les auteurs de [28] et [37] en 2000 et 2001. Leur étude montre que la plupart des systèmes d'exploitation utilisés sur des mobiles manquent de fonctionnalités importantes telles que : un support multi-utilisateurs, un contrôle d'accès au système de fichier basé sur des permissions et une protection mémoire. Les deux études pointent du doigt qu'aucun système d'exploitation n'est sécurisable à cause de l'absence de ces mécanismes basiques de sécurité (Linux est la seule exception).

Deux autres études furent menées pour améliorer la sécurité de Familiar-Linux, qui est une version de Linux pour les appareils mobiles.

La première étude [72] porta SELinux (Security Enhanced Linux) pour Familier-Linux. Cependant, comme SELinux fut développé à l'origine pour les serveurs, il ne se comporta pas trop bien sur les appareils mobiles, même après avoir supprimé certaines fonctionnalités.

La seconde étude [73] créa le système de sécurité Umbrella, qui fut spécialement conçu pour les appareils mobiles. Umbrella est un framework de protection qui utilise la notion de privilèges réduits plutôt que l'Access Control List et est basé sur un mécanisme de Mandatory Access Control. De plus, Umbrella implémente une sorte de *sandbox*, où un processus père peut contrôler les privilèges de ses processus fils.

7.2 Sécurité des mobiles

Des études sur la sécurité des mobiles et des smartphones ont analysé différents aspects de ces appareils. La plupart des travaux analysèrent le Bluetooth, le Short Message Service (SMS) et le Wireless Application Protocol (WAP).

Les problèmes de sécurités des mobiles liés au Bluetooth furent investigués par [74] et montrèrent de multiples problèmes sur plusieurs appareils différents. La plupart des problèmes découverts sont liés à des erreurs de logiques. Un exemple particulier est une vulnérabilité trouvée dans une application qui reçoit des fichiers envoyé via Bluetooth. Au lieu d'envoyer un fichier à l'application, un attaquant pouvait tromper cette application et avoir accès en lecture à une partie du système de fichiers. L'application pouvait alors envoyer un fichier quelconque situé sur le système de fichier en utilisant la connexion.

D'autres problèmes de sécurité sont liés au Short Message Service (SMS). Des études furent menées [75, 76, 77] sur différents modèles de mobiles et révélèrent des problèmes dans la gestion des messages SMS binaire. Les études montrèrent qu'en envoyant un message SMS mal formé, il était possible de geler le mobile ou de le redémarrer, menant à des attaques par déni de services.

D'autres études de sécurités se penchèrent sur l'infrastructure WAP. Deux études particulières [78, 79] démontrèrent l'efficacité du *fuzzing* dans l'analyse de sécurité des appareils mobiles et des composantes de l'infrastructure.

7.3 Programmes malveillants

Les programmes malveillants (malware) comme les virus, vers et chevaux de Troie sont devenus largement répandus durant ces dernières années, ciblant tous les systèmes d'exploitation courants. Le nombre de logiciels malveillants ciblant un OS spécifique est généralement proportionnel au succès de cet OS (exception faite de Linux, qui ne possède pas encore de programmes malveillants connus ciblant l'implantation de Linux pour appareils mobiles).

Symbian, l'OS le plus répandu, fait face à de multiples virus [80], vers [81] et chevaux de Troie [82, 83].

PalmOS est principalement la cible de virus et chevaux de Troie [84, 85, 86].

Windows CE/PocketPC est encore assez épargné par les virus. Notons cependant l'existence du virus [87] et du cheval de Troie [88].

7.4 Sécurité des infrastructures

Des études antérieures ont été menées sur plusieurs parties du réseau téléphonique mobile. Celles-ci ont principalement couvert trois catégories : l'infrastructure GPRS, le Short Message Service (SMS) et le Wireless Application Protocol (WAP).

Le réseau GPRS fut analysé dans [89], où certains problèmes de sécurité furent discutés. Un des problèmes était lié aux réseaux privés virtuels (Virtual Private Networks, VPN) employés par le GPRS ; ils sont seulement sécurisés par le fait de garder leur nom de point d'accès (Access Point Name) secret. Par conséquent un attaquant peut y avoir accès en devinant simplement l'APN. D'autres problèmes sont causés par l'infrastructure du GPRS qui repose sur un backbone basé sur UDP.

Une autre étude [90] a montré une attaque possible par surfacturation contre des utilisateurs du GPRS. L'attaque exploitait le manque de synchronisation entre deux parties de l'infrastructure GPRS et utilisait l'inondation de traffic pour augmenter les frais de la victime. Cette vulnérabilité fut rapidement réparée après sa découverte.

Une étude sur la sécurité de l'infrastructure SMS [91] révéla que des messages SMS envoyés depuis Internet pouvaient être utilisés pour effectuer une attaque distribuée par déni de service envers l'infrastructure de télécommunication mobile d'une grande ville. L'attaque augmente les délais de livraison des messages pour surcharger le réseau.

De multiples études [78, 79, 92] furent conduites sur les composants de l'architectures WAP. Ceux-ci montrent que pas mal de passerelles n'étaient pas capable de gérer certains champs d'en-têtes trop grands contenus dans les messages de réponses des serveurs HTTP. Un autre problème est lié à la nature UDP du système WAP, qui autorise des attaques par déni de services envers d'autres utilisateurs présents sur le réseau du même opérateur. Plus précisément, un attaquant peut usurper une requête WAP GET (un simple paquet UDP) à laquelle une passerelle WAP va répondre avec une réponse considérable (de multiples gros paquets UDP). La réponse sera envoyée à la victime, saturant son lien de connexion.

Une étude récente du BlackBerry [93], révéla plusieurs problèmes de sécurité liés à l'appareil et à l'infrastructure réseau spécialisée requise. Une des

vulnérabilités découvertes par les auteurs autorise un attaquant à éteindre le service BlackBerry d'une organisation entière. Cela montre que l'infrastructure utilisée par les appareils mobiles est une partie vitale de leur sécurité.

Chapitre 8

Conclusion

De nombreux problèmes que l'on retrouve sur les systèmes bureautiques commencent à apparaître sur les appareils mobiles.

Les différences architecturales entre ces appareils mobiles et les ordinateurs personnels (moins de mémoire, processeur plus lent, ...) constituent des défis pour les concepteurs en sécurité. L'infrastructure spécialisée sur laquelle reposent les mobiles accroît encore la difficulté et le temps nécessaire au développement et à l'analyse de sécurité.

Je souhaite qu'au travers des premiers chapitres introductifs le lecteur aura obtenu une meilleure connaissance de ces appareils ; qu'au travers du chapitre relatif à Java ME et Windows CE celui-ci aura saisi les dangers, mais aussi les différences, qui peuvent apparaître suite à l'implantation de logiciels et de fonctionnalités présents dans les ordinateurs personnels sur les appareils mobiles et enfin qu'au travers du chapitre sur les attaques croisées, le lecteur aura perçu les nouveaux dangers spécifiques dont doivent et devront faire face les smartphones à l'avenir.

J'estime qu'il y a un besoin grandissant d'outils efficaces permettant de tester la sécurité des appareils mobiles et des composants des réseaux mobiles ainsi qu'un besoin considérable de documentations et d'analyses de sécurité plus nombreuses dans le domaines de ces appareils.

De plus, l'augmentation spectaculaire à venir des smartphones ne pourra qu'accentuer l'apparition de nouvelles attaques et vulnérabilités et j'estime que des attaques, comme celles par services croisés présentées au chapitre six, deviendront courantes dans les années futures. Il y a un réel besoin d'adopter rapidement les mécanismes de sécurité de nos ordinateurs personnels pour ces appareils mobiles, comme des logiciels antivirus, antispyware (logiciel

espion) mais également des pares-feu et des mécanismes tels que le système d'étiquetage présentés dans ce mémoire qui n'existe encore que sur le papier.

Les analystes prévoient une augmentation spectaculaire du nombre de smartphones sur le marché dans les années à venir, ceux-ci pourront même à terme dépasser en nombre les ordinateurs personnels, de par leur nature mono-utilisateurs. Il est évident qu'à ce moment là, ils constitueront une cible de choix pour les attaques, ceci encore accentué par la nature payante de certains services associés. Néanmoins nous ne sommes qu'à l'aube de cette prolifération, ce qui nous laisse encore le temps de développer des systèmes et des mécanismes de sécurité adéquats pour s'assurer un avenir plus sûr.

Table des figures

Liste des tableaux

Bibliographie

[1] GSM World - Informations et spécifications des mobiles 2G et 3G. http://www.gsmworld.com/index.shtml.

[2] Two Billion GSM Customers Worldwide. http://www.prnewswire.com/cgi-bin/stories.pl?ACCT=109&STORY=/www/story/06-13-2006/0004379206&EDATE=.

[3] Comprehensive information about GPRS and Edge. http://www.3g-generation.com/gprs_and_edge.htm.

[4] Ghislain Bocq. Réseaux publics de télécommunication (ELEC321). Décembre 2006.

[5] CDMA Development Group (EVDO). http://www.cdg.org/.

[6] EVDO Info & Coverage. http://www.evdoinfo.com/.

[7] The Tech FAQ. http://www.tech-faq.com/.

[8] UMTS Forum. http://www.umts-forum.org/.

[9] 3rd Generation Partnership Project 2, 3GPP2. http://www.3gpp2.org/.

[10] WiMAX Forum. http://www.wimaxforum.org/home/.

[11] Portail du WiMAX 802.16. http://www.wimax-fr.com/.

[12] WiMAX - Norme 802.16. http://www.commentcamarche.net/wimax/wimax-intro.php3.

[13] DECT Forum. http://www.dect.org/.

[14] What is DECT ? http://searchnetworking.techtarget.com/sDefinition/0,290660,sid7_gci213889,00.html.

[15] Adam Stubblefield, John Ioannidis et Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. http://ftp.die.net/mirror/papers/802.11/wep_attack.pdf. 6 Août 2001

[16] Guy Leduc. Réseaux 1 (INFO 077). Décembre 2005.

[17] Andrew Tanenbaum. *Réseau*. Pearson Education 4e édition. Paris. 2003.

[18] IEEE 802 LAN/MAN Standards Committee. http://www.ieee802.org/.

[19] The Bluetooth Technology Info site. http://www.bluetooth.com/bluetooth/.

[20] Mindi McDowell et Matt Lytle. US-CERT National Cyber Alert System, Understanding Bluetooth Technology. http://www.us-cert.gov/cas/tips/ST05-015pr.html. 2005.

[21] Infrared Data Association. http://www.irda.org/.

[22] Infrared Data Association Information. http://www.answers.com/topic/infrared-data-association.

[23] Ultra-Wideband Forum. http://www.uwbforum.org/.

[24] Intel Corporation. Ultra-Wideband (UWB) Technology. http://www.intel.com/technology/comms/uwb/.

[25] Rafael Kolic. An introduction to Ultra Wideband (UWB) Wireless. http://www.deviceforge.com/articles/AT8171287040.html. 24 février 2004.

[26] Didier Godart. *Sécurité informatique : Risques, Stratégies & Solutions*. Edipro. 2005.

[27] Rick Broida. A guide to handheld operating systems. http://reviews.zdnet.co.uk/software/os/0,1000001098,39280712-1,00.htm. 10 août 2006.

[28] Arto Kettula, Security Comparison of Mobile OSes. http://www.tml.tkk.fi/Opinnot/Tik-110.501/2000/papers/kettula.pdf. 2000.

[29] Symbian OS - the mobile operating system. http://www.symbian.com/.

[30] Application signing and verification process for Symbian OS third party applications. https://www.symbiansigned.com.

[31] Symbian Developer Network, OS & Library Documentation. http://developer.symbian.com/main/oslibrary/.

[32] ACCESS (Palm OS). http://www.access-company.com/home.html.

[33] The History of Microsoft Windows CE. http://www.hpcfactor.com/support/windowsce/. 18 février 2007.

[34] Research In Motion. http://www.rim.net/.

[35] BlackBerry. http://www.blackberry.com/.

[36] Bryan Morgan. BlackBerry Operating System : the next monster ? http://www.wirelessinternetmag.com/news/0106/0106_devbiz_blackberry.htm. Juin 2001.

[37] Jukka Ahonen. PDA OS Security : Application Execution. Seminar on Network Security 2001. http://www.tml.tkk.fi/Studies/T-110.

501/2001/papers/jukka.ahonen.pdf.

[38] E. Rukzio. Application development with J2ME. Development of Interactive Applications for Mobile Devices. http://www.medien.ifi.lmu.de/diamd05/slides/j2me.pdf.

[39] M. Rohs. Application development with Symbian. Development of Interactive Applications for Mobile Devices. http://www.medien.ifi.lmu.de/diamd05/slides/symbian.pdf.

[40] J. Hamard. Mobile Human-Computer Interaction. Development of Interactive Applications for Mobile Devices. http://www.medien.ifi.lmu.de/diamd05/slides/mobilehci.pdf.

[41] ARM. www.arm.com.

[42] Mindi McDowell. Understanding Denial-of-Service Attacks. http://www.us-cert.gov/cas/tips/ST04-015.html. 2004.

[43] Jayan Krishnaswami. Denial-of-Service Attacks on Battery-Powered Mobile Computers. http://scholar.lib.vt.edu/theses/available/etd-02192004-003948/unrestricted/Jayan_Krishnaswami_Thesis.pdf. 13 Février 2004.

[44] Mark J. Handley et Eric Rescorla. Internet Denial-of-Service Considerations, RFC4732. http://tools.ietf.org/html/rfc4732. Novembre 2006.

[45] CERT Coordination Center. Denial of Service Attacks. http://www.cert.org/tech_tips/denial_of_service.html.

[46] Symantec. Threats to PDAs. http://www.symantec.com/avcenter/reference/malicious.threats.to.pdas.html. 2000.

[47] John Muir. Decoding Mobile Device Security. http://www.computerworld.com/securitytopics/security/story/0,10801,82890,00.html. 14 juillet 2003.

[48] Carnegie Mellon. Are Mobile Devices Safe ? http://www.mysecurecyberspace.com/articles/feature/are-mobile-devices-safe-.html. 2007.

[49] Tanya Candia. Mobile Phone Viruses : What Have We Learned ? http://www.technewsworld.com/story/47545.html. 14 Décembre 2005.

[50] SearchSecurity.com. Web application attacks Learning Guide. http://searchsecurity.techtarget.com/searchSecurity/downloads/WebappattacksLG.pdf.

[51] Wen-Chen Hu, Jyh-haw Yeh, Hung-Ju Chu et Chung-wei Lee. Internet-Enabled Mobile Handheld Devices for Mobile Commerce. *Contemporary Management Research* Vol 01. Page 13-34. http://www.cmr-journal.org/article/view/70/30. Septembre 2005.

[52] Peeter Paal. Java 2 Platform Micro Edition. http://www.tml.tkk.fi/Opinnot/Tik-110.501/2000/papers/paal.pdf. 2000.

[53] Kimmo Raatikainen. Java 2 Micro Edition Part 1. http://www.cs.helsinki.fi/u/kraatika/Courses/AMW02s/j2me.pdf. 14 mars 2002.

[54] Mourad debbabi, Mohamed Saleh, Chamseddine Talhi et Sami Zhioua. Java for Mobile Devices : A Security Study. http://www.acsac.org/2005/papers/151.pdf.

[55] Kent Inge Simonsen, Vebjørn Moen et Kjell Jørgen Hole. Attack on Sun's MIDP Reference Implementation of SSL. https://bora.uib.no/bitstream/1956/1901/3/Paper+5.pdf. 2006.

[56] André N. Klingsheim, Vebjørn Moen et Kjell J. Hole. Secure Networked J2ME Applications : Problems and Challenges. https://bora.uib.no/bitstream/1956/1901/2/Paper+6.pdf. 2006.

[57] Sun. Trusted MIDlet Suites using X.509 PKI. http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/midlet/

doc-files/PKITrust.html.

[58] Pierre Ficheux. Construction d'un système LINUX embarqué. http://docs.mandragor.org/files/Misc/GLFM/lm22/construction_sys_linux_embedded.html. Septembre 2000.

[59] Linux Phone Standards (LiPS). www.lipsforum.org.

[60] Open Source Development Labs (OSDL). www.osdl.org/lab_activities/mobile_linux.

[61] MontaVista - Linux-based Software and Development Tools for Connected Devices. http://www.mvista.com/.

[62] Mobilinux - Commercial Linux development platform. http://www.mvista.com/product_detail_mob.php.

[63] Tim Hurman. Exploring Windows CE Shellcode. http://www.pentest.co.uk/documents/exploringwce/exploring_wce_shellcode.html. 2005.

[64] Collin Mulliner. Exploiting PocketPC. http://www.cs.ucsb.edu/~seclab/projects/smartphones/2005_mulliner_WhatTheHack_slides.pdf. 2005.

[65] Stéphane Milani. Brève sur le GPRS et les attaques Overbilling entre autre. http://www.hsc.fr/ressources/breves/GPRSoverbilling.html.fr. 11 juillet 2006

[66] Sun Developer Network. Informations et documentations sur les librairies java. http://developers.sun.com/techtopics/mobility/.

[67] Nicolas FERRE. GNU/Linux comme système d'exploitation embarqué ? http://nferre.free.fr/emlnx/rapport/node8.html.

[68] Pierre Ficheux, *Linux embarqué*. Eyrolles, collection Blanche. 2e édition. Septembre 2005

[69] Microsoft. Windows CE Architecture. http://msdn2.microsoft.com/en-us/library/ms905093.aspx.

[70] C. Mulliner, G. Vigna, D. Dagon et W. Lee. Using Labeling to Prevent Cross-Service Attacks Against Smart Phones. *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment* (DIMVA) (pages 91-108).

[71] Mourad Debbabi, Mohamed Saleh, Chamseddine Talhi et Sami Zhioua. *Embedded Java Security*. Springer London. Première édition. Décembre 2006

[72] R. Coker. Porting NSA Security Enhanced Linux to Hand-held devices. http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Coker-OLS2003.pdf. 2003.

[73] Søren Nøhr Christensen, Kristian Sørensen et Michel Thrysøe. The Umbrella System. http://umbrella.sourceforge.net/umbrella_doc.pdf.

[74] M. Herfurt, M. Holtmann, A. Laurie, C. Mulliner, T. Hurman, M. Rowe, K. Finisterre et J. Wright. The trifinite group. http://www.trifinite.org.

[75] J. de Haas. Mobile Security : SMS and a little WAP. http://www.itsx.com/hal2001/hal2001-itsx.ppt.

[76] B. Jurry. Siemens Mobile SMS Exceptional Character Vulnerability. http://www.xfocus.org/advisories/200201/2.html. Janvier 2002.

[77] O. Whitehouse stack Inc. Nokia Phones Vulnerable to DoS Attacks . http://www.infoworld.com/article/03/02/26/HNnokiados_1.html. 26 février 2003.

[78] O. Whitehouse stack Inc. FuzzServer. http://www.blackops.cn/tools/FuzzerServer.zip.

[79] Oulu University Secure Programming Group. PROTOS Security Testing of Protocol Implementations. http://www.ee.oulu.fi/research/

ouspg/protos/. Janvier 2005.

[80] F-Secure. F-Secure Virus Descriptions : Commwarrior.A. http://www.f-secure.com/v-descs/commwarrior.shtml. 2005.

[81] Symantec Security Response. SymbOS.Cabir. http://securityresponse.symantec.com/avcenter/venc/data/epoc.cabir.html. Juin 2004.

[82] Symbian. Information about Mosquitos Trojan. http://www.symbian.com/news/pr/2004/pr20042561.html. 2004.

[83] F-Secure. F-Secure Virus Descriptions : Skulls. http://www.f-secure.com/v-descs/skulls.shtml. 2004.

[84] F-Secure. PalmOS/Vapor. http://www.f-secure.com/v-descs/vapor.shtml. Septembre 2000.

[85] McAfee. PalmOS/LibertyCrack. http://vil.nai.com/vil/content/v_98801.htm.

[86] McAfee. PalmOS/Phage.963. http://vil.nai.com/vil/content/v_98836.htm.

[87] Ratter/29A, C. Peikari et S. Fogie. WinCE4.Dust. http://www.informit.com/articles/article.asp?p=337069. Septembre 2004.

[88] Symantec Inc. Backdoor.Brador.A. http://www.symantec.com/avcenter/venc/data/backdoor.brador.a.html.

[89] FX of Phenoelit. More Embedded Systems. *In Defcon 11.* Août 2002.

[90] E. Gauthier. GPRS Overbilling Attack Using Unclosed Connections. Février 2003.

[91] P. M. W. Enck, P. Traynor et T. L. Porta. Exploiting Open Functionality in SMS-Capable Cellular Networks. *In Conference on Computer and Communications Security.* 2005.

http://www.smsanalysis.org/smsanalysis.pdf.

[92] FX and FtR of Phenoelit. Attacking Embedded Systems. *In Chaos Communication Camp*. Août 2003.

[93] FX of Phenoelit. Analyzing Complex Systems : The BlackBerry Case. *In BlackHat Briefings Europe*. Février 2006.

Annexe A

Code Java

```
public class rmsTest extends javax.microedition.midlet.MIDlet implements
    CommandListener {
    private Command exitCommand;
    private TextBox tb;
    static private SecurityToken classToken;
    public rmsTest(){
        exitCommand = new Command("Exit",Command.EXIT, 1);
        tb = new TextBox("Hello Midlet","hello world", 15, 0);
        Display.getDisplay(this).setCurrent(tb);
        tb.addCommand(exitCommand);
        tb.setCommandListener(this);
        RecordStore rs;
        try{
            rs = RecordStore.openRecordStore("attack", true);
        }
        catch (Exception e){ }
    }
    public void startApp() { }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable d){
        if (c == exitCommand){
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

Listing A.1 – Premier Midlet créant un enregistrement.

```
public class rmsMidlet extends javax.microedition.midlet.MIDlet implements
     CommandListener {
    private Command exitCommand;
    private TextBox tb;
    public rmsMidlet(){
        exitCommand = new Command("Exit",Command.EXIT, 1);
        tb = new TextBox("Hello Midlet", "hello world",1500, 0);
        Display.getDisplay(this).setCurrent(tb);
        tb.addCommand(exitCommand);
        tb.setCommandListener(this);
        String s = RecordStoreFile.getUniqueIdPath("Unknown", "rmsTest", "
            attack");
        boolean b = RecordStoreFile.deleteFile(s);
        if (b){
            tb.insert(" Deleting successful", tb.size());
        }
        else{
            tb.insert(" cannot delete", tb.size());
        }
    }
    public void startApp() { }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable d){
        if (c == exitCommand){
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

Listing A.2 – Deuxième MIDlet effaçant l'enregistrement du premier.

Annexe B

Shellcode sur architecture ARM

SUB R1, PC, #4 // R1 contient l'adresse de la prochaine instruction

FIG. B.1 – Code pouvant se localiser.

Si l'on désire coder *if ((a == b) && (c == d)) e++;*
Après avoir mis *a* dans R0, *b* dans R1, *c* dans R2, *d* dans R3 et *e* dans R4, on pourra écrire :
CMP R0, R1 // a==b
CMPEQ R2, R3 // c==d si a==b
ADDEQ R4, R4, #1 // e++ si a==b et c==d

FIG. B.2 – Exemple d'une suite d'instructions conditionnelles.

MOV R0, R0 // Inclut des zéros en binaire
MOV R1, R1 // N'inclut pas de zéros en binaire

FIG. B.3 – Exemple d'instruction remplaçant un NOP.

DLL	Fonction	h6315 (WinCE 4.2)	PDA2k (WinCE 4.21)
coredll	Sleep	0x01F713C4	0x01F71734
coredll	LoadLibraryW	0x01F71FF8	0x01F72368
coredll	MessageBoxW	0x01F89800	0x01F89CA0
coredll	fopen	0x01F9F0B0	0x01F9F688
coredll	fwrite	0x01FA5DD8	0x1FA63BC
winsock	socket	0x03691138	0x03071138
winsock	connect	0x03691148	0x03071148
winsock	recv	0x03691190	0x03071190
phone	PhoneMakeCall	0x03DE1270	0x03A01270

TAB. B.1 – Table des adresses des fonctions DLL.