SMARTPHONE SECURITY EVALUATION
The Malware Attack Case

Alexios Mylonas, Stelios Dritsas, Bill Tsoumas and Dimitris Gritzalis
Information Security and Critical Infrastrucutre Protection Research Laboratory
Department of Informatics, Athens University of Economics & Business (AUEB)
76 Patission Ave., GR-10434, Athens, Greece

Keywords: Smartphone, Security, Malware, Attack, Evaluation criteria.

Abstract: The adoption of smartphones, devices transforming from simple communication devices to 'smart' and multipurpose devices, is constantly increasing. Amongst the main reasons are their small size, their enhanced functionality and their ability to host many useful and attractive applications. However, this vast use of mobile platforms makes them an attractive target for conducting privacy and security attacks. This scenario increases the risk introduced by these attacks for personal mobile devices, given that the use of smartphones as business tools may extend the perimeter of an organization's IT infrastructure. Furthermore, smartphone platforms provide application developers with rich capabilities, which can be used to compromise the security and privacy of the device holder and her environment (private and/or organizational). This paper examines the feasibility of malware development in smartphone platforms by average programmers that have access to the official tools and programming libraries provided by smartphone platforms. Towards this direction in this paper we initially propose specific evaluation criteria assessing the security level of the well-known smartphone platforms (i.e. Android, BlackBerry, Apple iOS, Symbian, Windows Mobile), in terms of the development of malware. In the sequel, we provide a comparative analysis, based on a proof of concept study, in which the implementation and distribution of a location tracking malware is attempted. Our study has proven that, under circumstances, all smartphone platforms could be used by average developers as privacy attack vectors, harvesting data from the device without the users knowledge and consent.

1 INTRODUCTION

Smartphones are some of the devices that enhance Weiser's vision of ubiquitous computing (Weiser, 1991). Their small size, mobility, connectivity capabilities, and multi-purpose use are some of the reasons for their vast pervasiveness (Gartner, 2010).

Malicious software, or malware (Andleman, 1990; Cohen, 1989; Kephart & White, 1991) has also appeared in smartphone platforms (Hypponen, 2006), but initially their occurrences and severity were limited. Nonetheless, recent reports show that the risk of malicious application execution on smartphones is severe and contingent (McAfee, 2010; CISCO, 2011). Moreover, the use of smartphones extends the infrastructure perimeter of an organization, thus amplifying the impact and the risk of potential execution of malicious applications (Sindhu et al., 2010).

Apart from the increasing smartphone sales (Gartner, 2010), the annual downloads for applications developed for smartphones and distributed from official application repositories are also bound to increase by 117% in 2011 (Gartner, 2011). In addition, popular web applications (Gmail, YouTube, etc.) and social networks (Facebook, Twitter, etc.), are being accessed on mobile devices through native applications, instead of their usual web browser interface. In this context, smartphones contain a vast amount of the user's data, thus posing a serious privacy threat vector (PAMPAS, 2011; ENISA, 2011; GSMA, 2011). These data are augmented with smartphone sensor data (i.e. GPS) and data created by daily use (personal or business) making the device a great source of data related with the smartphone owner. This data source is invaluable to attackers trying to harvest them to increase their revenues (e.g., with blackmail, phishing, surveillance attacks). Hence, attackers try to infect smartphones with malware applications, harvesting

Mylonas A., Dritsas S., Tsoumas B. and Gritzalis D..
SMARTPHONE SECURITY EVALUATION - The Malware Attack Case.
DOI: 10.5220/0003446800250036
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2011), pages 25-36
ISBN: 978-989-8425-71-3

smartphone data without the user's knowledge and consent. It should be noted that the everyday use of smartphones by non-technical and non-security savvy people increases the likelihood of using smartphones as a security and privacy attack vector.

The security model of smartphone platforms has, under these circumstances, two contradicting goals. On the one hand, it must provide mechanisms to protect the users from attacks and on the other hand, it must attract third party developers, since the popularity of a platform depends on the attractiveness of its applications. The former goal is approached by each smartphone platform under a non unified and standardised approach that its effectiveness is controversial (Sophos, 2011). For the latter smartphone platforms provide developers with development friendly environments that include extensive documentation, programming libraries, and emulators. However, this development friendliness may also be used to write applications that can compromise the security and privacy of smartphone users more easily.

This paper examines the feasibility of malware development on smartphones by average programmers that have access to the official tools and programming libraries provided by smartphone platforms. This is mainly achieved through a proof of concept study that aims on evaluating the ease of malware development against users of smartphone devices. Thus, issues like state of the art attacks that might be performed by sophisticated attackers (Seriot, 2010; Lineberry et al., 2010) and the relation between malware attacks on smartphones and desktop computing devices are out of the scope of this paper.

This paper contributes towards this direction by (a) proposing a set of evaluation criteria, assessing the development of malware, and (b) providing a comparative case study analysis where the implementation and distribution of proof of concept location tracking malware is implemented in the current smartphone platforms.

The paper is organized as follows. Section 2 provides background information about current smartphone operating systems. In section 3 the comparative criteria are introduced, while in section 4 our privacy attack implementation scenarios are presented. Finally, the paper concludes in section 5.

2 SMARTPHONE PLATFORMS

In this section we discuss the security models and development environments of the surveyed smartphone platforms: (a). Android OS, (b). BlackBerry OS, (c). Symbian OS, (d). Apple iOS, and (e). Windows Mobile 6 OS. Our analysis focuses on application installation and execution. Security mechanisms that are used for the physical protection of the device (data encryption, anti-theft solutions, etc.) are out of the scope of this paper.

2.1 Android OS

The Android OS is a Linux based open source operating system developed and maintained by Google. Android was designed to be executed on portable devices, such as smartphones and tablets. It provides a free and publicly available Software Development Kit (SDK) that consists of tools, documentation and emulators necessary for the development of new applications in Java. According to Gartner (Gartner, 2010), the Android platform increased its worldwide smartphone sales from 3.5% in the Q3 of 2009 to 25.5% in the Q3 of 2010.

A core element of the Android security model (Google, 2011c) is the *manifest file*. The manifest file is bundled into the Android installation package file (.apk file), along with the applications bytecode and other related resources. The file follows the XML structure and provides all the necessary information to the Android platform for the execution of the application. Security-wise the manifest file is crucial for the system, since a developer defines in it the *permissions* of each application. These *permissions* control: (a) the way the application interacts with the system via access to system API, and (b) the way the system and the other applications interact with the given application's components. By default, every application runs in a sandboxed environment without any permission to perform an action that can impact the operating system itself, the other applications and the user.

Every application requests authorization for its permissions at installation time, which is based on the user's approval. No further checks are made during the applications' execution. Hence, if the user decides to grant permission to the application, then the protected system resources are available to the application, otherwise the access to the resources is blocked

The installation package file of every Android application has to be digitally signed by its developer. Android's security model then maps the signature of the developer with a unique ID of the application package and enforces signature level permission authorization (Google, 2011c). However, in the Android security model it is not obligatory that the developer's certificate is signed by a trusted

certificate authority. Thus, the applications are usually digitally signed with self-signed certificates, providing only poor source origin and integrity protection. This preserves the anonymity of a potential attacker, since the certificate is not verified by a Trusted Third Party (TTP).

A developer distributes her application either in the official Android application repository maintained by Google, the Android Market , or outside the repository. Google does not enforce any restriction in the installation of applications originating outside its repository. On the other hand, Google developed technologies to remove applications (Google, 2011a) from the devices and the Android Market in case they pose a threat to the Android platform. The applications in the Android Market are provided to end users without being tested for malicious behaviour. Hence, a developer must only provide her Google account and pay a small fee for the distribution of any application in the Android application repository. It is evident that this procedure is not able to stop malware from being distributed via the Android Market and being installed on devices.

According to (Google, 2011b), version 2.2 is the dominant Android version, currently deployed in the majority of the Android devices (51.8%). Older versions of the OS are still being in use: 35,2% of the devices run version 2.1, while version 1.6 runs on 7.9% of the devices. The adoption of the latest version of the OS (version 2.3) is still low, since it is deployed only in the 0.4% of the Android devices.

2.2 BlackBerry OS

The BlackBerry OS is an operating system maintained by Research In Motion Inc. (RIM). The current version of the OS is version 6. The OS is executed on BlackBerry smartphones and tablet devices created by RIM. According to Gartner (2010), RIM's device worldwide market share dropped to 14.8% in the Q3 of 2010 from 20.7% in the Q3 of 2009.

Documentation about the OS details is not provided by RIM. However, the company provides, through the BlackBerry SDK, the related documentation, tools, API and emulators, which are necessary for application development.

The platform security model (RIM, 2011b) enforces restrictions to third party applications trying to access protected APIs of the OS, by demanding the signing of the application with a cryptographic key provided by RIM (RIM, 2011a). A developer needs to pay a small amount in order to acquire a valid RIM signing key pair. However, this process only provides poor source origin and code integrity and does not offer any assurance about the validity and/ or the security level of the third party application.

The official application repository for BlackBerry smartphones is the App World (RIM, 2011b). Unofficial repositories of applications, such as Crackberry (Smartphone Experts, 2011), also exist.

Application distribution in the official repository requires registration for a vendor account. Developers incur a registration cost of $200 for the aforementioned account creation, allowing them to submit 10 applications. For additional submissions an administration fee is required. It should be noted that before the application publication in the repository, its code is not examined by BlackBerry for malicious behaviour and, in addition, BlackBerry does not operate a remote application removal mechanism.

Finally, the platform provides the following options for the installation of applications in the device (a). via the BlackBerry App World, (b). Over the Air (OTA), (c). via the device's browser, and (d). via RIM desktop synchronization software.

2.3 Symbian OS

Symbian OS is an operating system maintained by Nokia. Symbian's current version is *Symbian^3*. The platform had a major worldwide market share decrease in Q3 of 2010 falling to 36.6% from 44.6% in the Q3 of 2009 (Gartner, 2010).

Symbian is executed in smartphones and provides multiple free and publicly available SDKs. The SDK includes the tools, documentation and emulators that are necessary for the development of new applications, written in C++.

The cornerstone in Symbian's security model is the use of *capabilities* (Nokia, 2011a) for defining restrictions to sensitive platform APIs. These capabilities are grouped in the following categories: (a). *basic*, (b). *extended*, (c). *manufacturer*. The first category includes basic functionality (e.g. access to the network, access to location data, etc.), where the user is prompted for its authorization during installation. The second capability category controls the use of sensitive API that is only granted through the Symbian Signed process (Nokia, 2011c). The last capability category controls application access to the most sensitive API in the platform (i.e. *All-Files*, *DRM*, *TCB*). These capabilities are only granted by a Device Manufacturer (e.g. Nokia, Sony Ericsson, etc). As indicated by Nokia (2011a) the *basic* capability category contains sufficient functionality for application development. In this context, our proof of concept study uses capabilities.

For each application installation, signing the application's package file *(.sis file*) is required by the Symbian security model. Signing ensures that the

application is not using API apart from the one corresponding to the applications signing level (Nokia, 2011a). If the application uses only *basic* capabilities the developer can self sign it (Nokia, 2011b). Self-signing has the advantages to be performed in the developer's computer, and it is not necessary to map the application installation package file with a device IMEI. This results in no restriction during multiple device installations, but the smartphone user will be prompted with security warnings at installation time, since the signing key is not trusted. To eliminate the warnings and access sensitive capabilities the developer submits her application to Symbian Signed along with the list of device IMEIs. However, guidelines for bypassing Symbian's secu-rety model are available (Symbian Freak, 2011), allowing the execution of unsigned applications, but the modified version of the OS is out of scope of this paper.

The applications are not required to reside in an application repository in order to be installed to Symbian devices. Nonetheless, Nokia maintains an official application repository, the OVI store.

2.4 iOS

iOS is a proprietary operating system maintained by Apple. iOS is only executed in Apple smartphones and tablets (i.e. iPhones, iPADs) and its current version is 4.2.1. According to Gartner (2010), Apple's worldwide smartphone market share dropped slightly to 16.7% in the Q3 of 2010 from 17.1% in the Q3 of 2009.

Apple provides, after registration to the company's Dev Center (Apple, 2011a), documentation, tools and the necessary API for application development in *Objective C*. It should be noted that the toolset provided by Apple is only compatible with Mac OS X operating system.

The official repository of iOS applications is the App Store. The distribution of applications in the repository costs $99 per year (Apple, 2011b). Nonetheless, the iOS protection code can be altered and bypassed (jailbroken) and the user is able to install applications that are not officially signed by Apple from an unofficial repository, such as *Cydia Store, Installus,* etc. Installation of applications in modified versions of the OS is out of the scope of this paper.

The security model of the iOS only permits the installation of applications that have been signed by Apple (Apple, 2011a). Before being signed, an application is tested for its functionality consistency and for malicious behaviour. However, the testing process and criteria applied by Apple are not publicly available.

Applications are installed on the device with: (a). the use of an application installed on the device that connects to Apple's App Store, or (b). the use of cross-platform synchronization desktop software provided by Apple (iTunes). Once the application is installed to the device the user neither controls nor is prompted when an application accesses some OS' sensitive resources. An in depth analysis of all the data that are available to an application in version 3 of iOS is provided in (Seriot, 2010).

2.5 Windows Mobile

Windows Mobile is a smartphone OS developed and maintained by Microsoft. The OS's worldwide smartphone market share decreased from 7.9% in the Q3 of 2009 to 2.8% in the Q3 of 2010 (Gartner, 2010). The latest version of the OS is Windows Phone 7. However, until the writing of this paper Microsoft has not made available basic API functionality, such as sockets for internet connectivity (Microsoft, 2010 d). Therefore, in this section we present the security model of Windows Mobile 6, since its development API is available.

The security model of Windows Mobile (Microsoft, 2010c) depends on the enabled policy of the device. This policy is responsible for controlling which applications are allowed to be executed on the device, what functionality of the OS is accessible to the application, how desktop applications interact with the smartphone, and how the user or application access specific device settings. The enabled policy on a Windows Mobile smartphone is either *one-tier access* or *two-tier access* (Microsoft, 2010c).

A device with one-tier access policy enabled, only controls if one application runs on the device or not, without inspecting if the application is using sensitive API. This decision depends on whether the application's installation package file (*.cab file*) is correctly signed with a certificate that exists in the device's certificate store. If the application is signed with a known certificate, then the application runs in privileged mode, with the ability to call any API, access and modify anything in the device's file system and registry. Otherwise, if the application is unsigned or signed with a certificate that is not known, further policy checks take place for the decision of application execution. In this case, security policies settings define whether the user is prompted to give her consent for the application to run. It must be clarified that if the user permits the execution, then the application will run in privileged mode. This means that an unknown and unsigned application maintains full access to the device.

On the other hand, a device with two-tier access policy enabled, apart from controlling application execution, it also checks runtime permissions by controlling the APIs that the application uses. Access to protected API is determined by the application's digital signature. More specifically, if the application is signed with a known certificate (i.e. a certificate present in the device's certificate store), then the application is executed without further checks, granted the permissions defined by the certificate class. In the case that the certificate belongs to the *Privileged Execution Trust Authorities* certificate store, the application is executed with privileged permissions. Otherwise, the application is executed in normal mode. When the application is unsigned or signed with an unknown certificate, then further checks are required to determine if the application is allowed to run in normal mode. It is worth noting that the functionality provided by normal privileges is enough for most third-party developed applications.

According to (Microsoft, 2010c; Microsoft, 2010 a) the default security configuration of Windows Mobile, provides weak security protection as: (a). it allows the execution of unsigned applications or singed ones with an unknown certificate, (b). in case (a). the user is prompted to authorize the execution of the application. Hence, in both access tiers of the default security configurations, unsigned and unknown code is executed with the user's approval either in normal mode (*two-tier access*) or privileged mode (*one-tier access*). Furthermore, although *one-tier access* does not provide strong security, it is the default access tier in some devices (Microsoft, 2010c). Nonetheless, it should be noted that the security model permits Mobile Operators to make post-production changes to security settings configurations of the device.

The security model of Windows Mobile includes security mechanisms enabling a Mobile Operator to revoke (i.e. remove) applications running on smartphones (Microsoft, 2010c). The revocation may concern either (a). a class of applications signed with the same certificate, where the corresponding certificate is being revoked, or (b). a specific application binary, where the hash of the binary is being transferred to the device with transfer mechanisms described in (Microsoft, 2010c).

For application implementation in Windows Mobile 6, Microsoft freely provides the required development toolkit (i.e. SDK, emulator, documentation, etc.). The supported implementation languages (e.g. C#, Visual C++) are compatible with the *Compact .NET Framework*.

For the acquisition of certificates that are known to the devices, the developer opts from the paid services provided by Microsoft (Microsoft, 2011b).

3 COMPARATIVE EVALUATION OF SMARTPHONE PLATFORMS

This section provides a comparative evaluation of the smartphone platforms in terms of malware development and distribution. Our analysis examines the feasibility of attacks implemented by average application developers. More specifically, the presented evaluation is based on: (a). the definition of qualitative evaluation criteria, and (b). a proof of concept malware implementation study, in which the development of a location tracking application is examined. At this point it should be stressed that any sophisticated attack conducted by experienced attackers, as well as, publicly available malicious code used by script kiddies are not examined in this paper. Furthermore, a comparison with malware development in desktop computing is not examined in this paper either.

3.1 Evaluation Criteria

The comparative evaluation of smartphones is performed by defining and using a set of evaluation criteria, which are elaborated in this section. The proposed criteria concern the development platform and the developer. From the proposed evaluation criteria the former are objective, relying solely on characteristics of the smartphone platform. The latter are subjective, giving details about the attack development effort and as a result depend on the developer's skills and background. The latter, however, are given as an indication on the effort needed to conduct such attacks via smartphone applications.

Our overall approach focuses primarily to the objective criteria (development platform), while at the same time takes into account the subjective criteria (regarding the developer side). It must be noted that this list of criteria is not exhaustive. Table 1 summarizes the proposed evaluation criteria.

3.1.1 Development Platform Criteria

In this section we describe and analyse the introduced development platform evaluation criteria in relation with their possible data type.

Development Tools Availability {Yes, Partial, No}: This criterion refers to the availability of deve-

lopment tools needed for application development. The public and free nature of these tools makes the development of malware easier and cost effective. The reason for this is that the presence of a free emulator reduces the development cost of the attacker, since a purchase of a device is not necessary. In addition, the SDK contains all the tools (e.g. debuggers, compilers, etc.), which are necessary for the implementation of the malicious application.

Development Friendliness {Yes, No}: This Boolean criterion assesses the "developer" friendliness of the programming language supported by the smartphone platform. The adoption of a well known and widely used programming language (e.g. Java) is preferred during any application deployment.

Installation Vectors {Multiple, Restricted}: This criterion assesses the available installation options for an application on the smartphone device. These installation options include the use of removable media, through the WEB, email, etc.

Application Portability {Yes, No}: This criterion refers to the ability of the malicious application to be executed in different versions of the target smartphone OS. The more compatible an application with different versions of the OS is, the greater the attack target population becomes.

Application Testing {Yes, No}: This criterion refers to the possible application testing procedures, which could be used from official vendors (e.g. Apple) in order to determine applications' potential malicious activity. The tests take place before the application is available in the official vendor application repository.

Application Removal {Yes, No}: This Boolean criterion refers to the existence of a remote application removal mechanism. The automated removal of an application from the repository and the smartphone devices is triggered when enough evidence is discovered that the application acts in a maliciously way.

Unofficial Repositories {Yes, No}: This Boolean criterion refers to the existence of application sources outside the official application repository. In the case that the official repository adopts application testing procedures, one option for a potential attacker is to place the application in alternative sources. This is a common action when the security model of the smartphone permits the installation of applications from sources other than the official repository.

Distribution Cost {Yes, No}: This criterion assesses whether the cost of application distribution into the official application repository deters a potential attacker.

API Restrictions {Yes, No}**:** This criterion refers to the restrictions imposed by smartphones' OS, in terms of how they control the use of protected APIs.

Application Signing {Yes, No}: This criterion is used for assessing the restrictions imposed by smartphones' OS, concerning the signing of the applications, before they are installed on a device.

3.1.2 Developer Criteria

This set of criteria includes the Developer's Background and the Development Time. In specific:

Developer's Background {Education Level}, refers to the developer's knowledge in information security as well as to her programming skills. We assume that the amount of knowledge a developer possesses in information security and her programming skills, determine the sophistication of the attacks she is able to implement.

Development Time {Number}, which is used for determining the effort needed to conduct a malware attack.

Apparently, the abovementioned criteria are given as an indication for the time and skills needed for the development of an attack by an average-skilled programmer. The evaluation criteria are summarized in Table 1.

Table 1: Proposed Evaluation Criteria.

Evaluation Criteria	Type
Development Tools Availability	String {Yes, Partial, No}
Development Friendliness	Boolean
Installation Vectors	String {Multiple, Restricted}
Application Portability	Boolean
Application Testing	Boolean
Application Removal	Boolean
Unofficial Repositories	Boolean
Distribution Cost	Boolean
API Restrictions	Boolean
Application Signing	Boolean
Developer's Background	Education Level
Development Time	Number

4 IMPLEMENTATION OF A MALWARE ATTACK

In this section the implementation of a malware attack is presented, as a proof of concept study. The

criteria defined in the previous section are applied to evaluate the robustness and the security properties of the smartphones platforms under examination.

Our study examines the implementation feasibility of a simple attack scenario (see Figure 1).

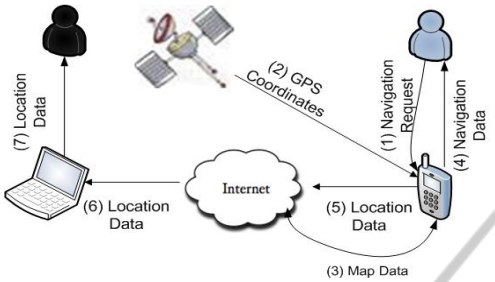

Figure 1: Case study attack scenario.

The attack scenario refers to a location tracking malicious application. The application collects the smartphone's user GPS coordinates (i.e. her exact position) and sends them to the attacker. It is assumed that the malicious functionality is included in a free GPS navigation application. The application apart from getting the user's location and presenting it using *Google Maps,* it also sends the location data to the attacker server. The described malware, in most cases, is executed without creating any suspicion to a naive smartphone user. The reason for this is that the application's requests (i.e. access to device location and Internet access to connect to the Internet) are consistent with the application's expected functionality.

We have implemented our development case study in our lab using two computers running a Windows XP and a Mac OS Leopard operating system. In the Windows machine we installed the emulators and the SDKs of all the smartphone platforms, apart from Apple's iOS that was only compatible with Mac OS X.

For the malware implementation we selected two computer science students (one undergraduate and one post-graduate student, respectively) with basic information security background and moderate programming skills. Before the case study implementation the students had successfully completed information security related courses that are consistent with the Common Body of Knowledge described in (Theoharidou et al., 2008). The undergraduate student had completed a course on Information Security Management and the postgraduate had completed the courses Information Security Management, Information System Auditing, Network Security, and Cryptography. Both students were more familiar with the Java programming language, since this was the language used in implementations during undergraduate and postgraduate projects. The post-graduate student would only be involved in the proof of concept attack implementation in a smartphone platform only if the undergraduate was unable to implement it.

In the following paragraphs we analyse the results regarding the development and use of this malicious application to the smartphone platforms described in Section 2 to check their security model robustness to our attack scenario. The platforms that we have examined are: Android OS, BlackBerry OS, Symbian OS, Apple's iOS and Windows Mobile 6 OS. The results of the case study are presented in the sequel.

4.1 Android OS Case Study

Our attack implementation was successfully developed on the Android platform in one day. The official development toolkit (i.e. SDK and emulator) was used for the implementation purposes. The reasons why our attack implementation was efficient are: a). the adoption by the platform of a widely used programming language (i.e. Java), and b). the effective documentation of its API. In addition, the same source code successfully compiled and executed in versions 2.1 and 2.2 of the Android platform; hence the application is considered portable within the Android platform at the time of the writing.

Regarding application distribution there are many options for the attacker. The reason for this is that the Android platform does not impose any restriction neither on the source of application (i.e. originating from an unofficial repository) nor on the installation vector (e.g. removable media, WWW, etc.). A small registration fee is required for the placement of applications in the official repository, but it is considered inadequate to impede an attacker. Furthermore, even if the official repository is selected for the distribution, application testing for malicious behaviour is not taking place. Hence, it is likely that malware such as one described in this case study is currently present and downloaded by naive users from the repository.

As we have already mentioned the security model of the Android platform imposes some application restrictions concerning the application signing and API control. We argue that these restrictions provide only partial security protection. For the former, API control restrictions are authorized by the naive user only during the application installation. No further checks about application permissions take place after the installation. Hence, it is likely

that the malicious application would be granted the requested permissions (i.e. access to location data and the Internet), especially in our case, where the permissions fully match the expected application's functionality. For the latter, the imposed signature can be self-signed by the developer and as a result the application's source origin is not verified. This situation, combined with the fact that an attacker may find valid credit cards numbers in the underground market, could be used to commence elite spoofing attacks. These attack scenarios are out of the scope of this paper. Finally, Google's remote removal mechanism is the only efficient post installation protection mechanism against our case study scenario.

From the above analysis we infer that the likelihood of conducting such an attack on the Android platform is very high. In this context, Table 2 summarizes the results of our case study based on the criteria we have defined in section 4.1.

Table 2: Android OS Analysis and Results.

Evaluation Criteria	Android OS
Development Tools Availability	✓
Development Friendliness	✓
Installation Vectors	multiple
Application Portability	✓
Application Testing	×
Application Removal	✓
Unofficial Repositories	✓
Distribution Cost	×
API Restrictions	✓
Application Signing	✓
Developer's Background	B.Sc.
Development Time	0.5day

4.2 BlackBerry OS Case study

Regarding our attack analysis on BlackBerry platform, the results were again successful. The attack was conducted by the B.Sc. student. RIM's official development toolkit (i.e. SDK and emulator) was used for attack implementation. The attack implementation was not demanding and its duration was one day. The reasons for the effectiveness of the attack implementation are the adoption by the platform of a widely used programming language (i.e. Java), and the effective documentation of its API. Furthermore, the same source code successfully compiled and executed in versions 5/6 of the BlackBerry platform, therefore the application is considered portable.

The security model of the BlackBerry does not impose any restrictions regarding the origin of the application. Nevertheless, the application package file (.cod) must be signed to access restricted and sensitive platform APIs. For the signing process the developer incurs a small key acquisition fee. As there is no strong authentication in the key acquisition process, the signing of the application provides only integrity protection and poor source origin.

The cost for the cryptographic keys is considered inadequate to deter an attacker. On the other hand, the cost for the publication is expected to impede an attacker from publishing the application in the RIM's official repository, especially if she is in possession of limited economic resources; the attacker still has the option to submit an application to an unofficial repository.

The security model of the RIM's platform does not employ any application testing before accepting the submission of an application. Furthermore, there is no application removal system automatically removing applications with malicious behaviour. Hence, if the malware application is submitted in the official repository, then it is very likely to be downloaded and installed in BlackBerry devices.

Conclusively, the development of the malware attack examined in this scenario is feasible and it demands little development effort. The only impediment is the cost of submitting the application to the official repository. Table 3, depicts the results of the case study regarding Blackberry OS.

Table 3: BlackBerry OS Analysis and Results.

Evaluation Criteria	BlackBerry OS
Development Tools Availability	✓
Development Friendliness	✓
Installation Vectors	multiple
Application Portability	✓
Application Testing	×
Application Removal	×
Unofficial Repositories	✓
Distribution Cost	✓
API Restrictions	✓
Application Signing	✓
Developer's Background	B.Sc.
Development Time	1 day

4.3 Symbian OS Case study

Symbian OS provides basic functionality sufficient for application development providing the developer the option to self sign her application. Nevertheless, some compatibility issues exist, since the *location capability* -which controls access to API determining the location of the device - does not reside in the *basic capability* category in some Symbian OS versions (Nokia, 2011a).

For the deployment of the malware attack scenario only the *basic* capability category was used. Self-signed applications create a security warning at installation time that the user has to accept. Even so, the user would likely accept the installation of the application bypassing and ignoring the security warnings. Apart for signing the application, the security model of the platform does not restrict the application's distribution and as a result the attacker has many distribution options (e.g. through an attacker-controlled application repository). Hence, the attacker has the option to avoid distribution cost and the Symbian Signed application testing. In addition, Symbian will only be able to revoke the self-signed certificate used in this case study, if Symbian becomes aware of the malware binary existence.

The attack implementation was not successfully completed in the Symbian platform. Even though the implementation was performed with the officially recommended development toolkits, the case study developers were unable to compile their code. In addition, even the sample applications provided by Symbian could not be compiled. The installation of the development toolkit was fully automated and the developers did not participate in its configuration. Hence, the possibility of toolkit misconfiguration is eliminated.

Table 4: Symbian OS Analysis and Results.

Evaluation Criteria	Symbian OS
Development Tools Availability	✓
Development Friendliness	×
Installation Vectors	Multiple
Application Portability	×
Application Testing	×
Application Removal	✓
Unofficial Repositories	✓
Distribution Cost	×
API Restrictions	✓
Application Signing	✓
Developer's Background	M.Sc.
Development Time	N/A

Furthermore, the developers faced other development obstacles during our case study, namely inadequate structure in the platform's API documentation (e.g. encountered "file not found" links), and the fact that they were not familiar with the platform's programming language. However, the obstacles reported in this subsection are of a minor importance to an experienced attacker; a case which is out of scope in this paper. Hence, the above obstacles are likely to deter unmotivated attackers from developing malware attacks. The results of the case study on Symbian platform are presented in Table 4.

4.4 iOS Case study

The implementation of our simple location tracking attack was successfully completed on Apple's iOS. The implementation lasted seven 7 days and was tested on emulators running iOS 3 (version 3.1.2) and iOS 4 (version 4.1). The implementation duration was expected a priori to last more than in the other platforms due to no prior experience with *Objective C*. However, the toolset provided by Apple (i.e. SDK, documentation and emulator) minimized this lack of experience. In Table 5 the relevant criteria was assigned the value *partial,* since the provided toolkit is available only to Mac OS X users.

The installation of applications to devices running the iOS system is only possible via Apple's App Store. Hence, unofficial application repositories are not available for devices running the official version of iOS. As a result, a malware author must submit the application to the official repository. The submission of an application to the App Store requires a non free registration to Apple's development program. Prior to the inclusion of an application in the official repository it must be examined and signed by Apple. Nonetheless, the process of application testing (e.g. static binary code analysis, dynamic binary code analysis) is not available. The application testing criteria are also not available, apart from the rejection of not official Apple's API usage. This security mechanism may be circumvented by a sophisticated attacker using encrypted payloads and logic bombs on the binaries.

Table 5: iOS Analysis and Results.

Evaluation Criteria	Apple's iOS
Development Tools Availability	partial
Development Friendliness	×
Installation Vectors	restricted
Application Portability	✓
Application Testing	✓
Application Removal	✓
Unofficial Repositories	×
Distribution Cost	✓
API Restrictions	×
Application Signing	✓
Developer's Background	M.Sc.
Development Time	7days

The user has no control on the application's actions after the installation of an application in the device. In addition, the user is not informed when the application uploads data to a remote server. In our case study, the user would only be prompted to permit access to location data. But, as the applicati-

on is providing location based services the user is expected to confirm access to location data. Afterwards, the data would be transferred to the attacker's remote server, without the user noticing it.

Apple has developed security mechanisms allowing the remote deletion of malicious applications from its devices and their removal from the application repository. This is a significant post installation security mechanism in case Apple or the user became suspicious of the malware software. The results of the case study in iOS platform are depicted in Table 5.

4.5 Windows Mobile 6 Case Study

The implementation of our location tracking application on a Windows Mobile device was successfully completed by the student in 2 days. The programming language was *C#.* The reasons of the effectiveness of the attack implementation are the adoption by the platform of programming language, namely *C#* that resembles the programming rationale of Java and the effective documentation of its API.

The proof of concept malware software was implemented for the versions of Windows Mobile 6.1 and 6.5 using the SDK provided by Microsoft. The installation package of the application was not signed. During the implementation the default configuration of the security model was preserved, in essence that: (a). unsigned applications would be allowed to run, (b). the user would be prompted to authorize the application execution, and (c). if the application had been authorized by the user it would have full access to the smartphone's OS system services.

The security model of Windows Mobile does not impose restrictions on the installation vector of applications. Moreover, applications are able to be installed on the devices even if they are downloaded from a source outside Microsoft's official repository. Hence, the attacker does not have application distribution costs. Furthermore, the application is not being tested for malicious behaviour, since it is not distributed by Microsoft distribution services. Nonetheless, the application removal mechanism, applied by Microsoft, may be used for the automated removal of the implemented malware application.

To sum up, the feasibility of our malware attack in Windows Mobile depends on the device configurations regarding the security model, the user authorization at installation time and the automated application removal security mechanism. Table 6 summarizes the results, w.r.t. Windows Mobile platforms.

Table 6: Windows Mobile OS Analysis and Results.

Evaluation Criteria	Windows Mobile
Development Tools Availability	✓
Development Friendliness	✓
Installation Vectors	multiple
Application Portability	✓
Application Testing	×
Application Removal	✓
Unofficial Repositories	✓
Distribution Cost	×
API Restrictions	×
Application Signing	×
Developer's Background	B.Sc.
Development Time	2days

4.6 Case Study Overview

The security model of a smartphone operating system has two contradicting goals. On the one hand, it must provide users with security assurance concerning the execution of third-party applications on their devices. On the other hand, it must provide the developers with a secure system where on the one hand, consumers are willing to install new applications and on the other, it is easy and efficient to implement new applications.

The proof of concept exercise demonstrated that, under certain circumstances, the security model of all available smartphone platforms would not counter a location tracking attack. Moreover, it showed that it is possible to easily implement the attack on all smartphone platforms, except from Symbian and iOS.

The reasons of the implementation failure on Symbian were not security related. They were related with the developer's programming skills and Symbian's unstructured API documentation.

The implementation in all other platforms was efficiently and effectively completed by average programmers and tested on the officially provided emulators. Nonetheless, by using the API documentation most of the attacks were implemented by the undergraduate student. This is a serious indication of how malicious software can evolve in smartphones.

Application testing for malicious behaviour cannot be avoided only on Apple's iOS. Furthermore, iOS was the only platform having strict installation requirements.

Among the examined platforms only Windows Mobile allowed, under some security model configurations, the execution of unsigned applications. Yet, the digital signature process gives different security assurance to the smartphone holder on the examined platforms. The user was found, on the one hand, not having any control on the API running in the device,

Table 7: Malware scenario implementation overview.

Evaluation Criteria	Android	BlackBerry	Symbian	iOS	Windows Mobile
SDK & simulator availability	✓	✓	✓	partial	✓
Development Friendliness	✓	✓	×	×	✓
Installation Vectors	multiple	multiple	multiple	restricted	multiple
Application Portability	✓	✓	×	✓	✓
Application Testing	×	×	×	✓	×
Application Removal	✓	×	✓	✓	✓
Unofficial Repositories	✓	✓	✓	×	✓
Distribution Cost	×	✓	×	✓	×
API Restrictions	✓	✓	✓	×	×
Application Signing	✓	✓	✓	✓	×
Developer's Background	B.Sc.	B.Sc.	M.Sc.	M.Sc.	B.Sc.
Development Time	0.5day	1 day	N/A	7days	2days

on some platforms. On the other hand, the user is fully responsible for authorizing application execution on other platforms. The latter is identified as a major weakness in the security model of some smartphones platforms, since the user's security knowledge and awareness is often insufficient.

Only in two smartphone platforms an attacker could not avoid distribution costs and only four of them contained remote application removal mechanisms. Table 7 summarizes our findings.

According to the case study findings, a non proficient attacker would not choose to use the iOS as a privacy and security attack vector, since it is the platform having the most defensive mechanism in place (i.e. application testing, controlled application installation vectors and remote application removal) and being difficult in application development. An attacker is expected to use one of rest platforms, where, in this case study, Android and Windows Mobile were found to provide the least protection against our attack scenario.

5 CONCLUSIONS

Smartphone devices are multi-purpose portable devices enclosing a vast amount of third party applications that augment the device's functionality. The existing smartphone security models facilitate mechanisms and processes controlling the installation and execution of third party applications. Even so, the sufficiency of the adopted security mechanisms seems to be controversial. Their ability to protect the devices from being a privacy attack vector from developers, such as undergraduate and postgraduate computer science students, is proven to be unclear.

Our paper (a). proposes evaluation criteria assessing the development of smartphone malware, and (b). provides a comparative case study analysis where the implementation and distribution of proof of concept location tracking malware is attempted in the current smartphone platforms.

Our proof of concept study has proven that under circumstances all smartphone platforms would not stop average developers from using them as privacy attack vectors, harvesting data from the device without the user's knowledge and consent. It also showed the easiness of malware application development by average programmers that have access to the official tools and programming libraries provided by smartphone platforms.

A silver bullet solution against similar attack scenarios is not available. Some of the solutions that can be used to avoid a potential malware outbreak in smartphones are: (a) user awareness, i.e. informing user about security and privacy risks in smartphone platforms, and (b) providing secure application distribution in smartphone platforms.

In this context our further work will focus on providing a secure application distribution scheme for smartphone applications. Moreover, we plan to extend the evaluation criteria and attribute weights to them. We also plan to repeat the case study with more developers to acquire more generalizable results.

ACKNOWLEDGEMENTS

This work has been partially funded by the European Union (European Social Fund) and Greek national funds through the Operational Program Education

and Lifelong Learning of the National Strategic Reference Framework - Research Funding Program: HE-RACLEITUS II - Investing in Knowledge Society.

REFERENCES

Adleman, L. (1990). An Abstract Theory of Computer Viruses. In S. Goldwasser, *Advances in Cryptology — CRYPTO' 88* (pp. 354-374). Berlin: Springer/ Heidelberg.

Apple. (2011a). *iOS Dev Center*. Retrieved February 18, 2011, from http://developer.apple.com/devcenter/ios/

Apple. (2011b). *iOS Developer Program*. Retrieved February 18, 2011, from http://developer.apple.com/ programs/ios/

CISCO. (2011). *Cisco 2010 Annual Security Report*. Retrieved February 18, 2011, from http:// www.cisco.com/en/US/prod/vpndevc/annual_security _report.html

Cohen, F. (1989). Computational aspects of computer viruses. *Computers & Security*, 8(4), 325-344.

Gartner. (2010). *Gartner Press Releases*. Retrieved February 18, 2011, from http://www.gartner.com/it/ page.jsp?id=1466313

Gartner. (2011). *Gartner Press Releases*. Retrieved February 18, 2011, from http://www.gartner.com/it/ page.jsp?id=1529214

Google. (2011a). *Exercising Our Remote Application Removal Feature*. Retrieved February 18, 2011, from http://android-developers.blogspot.com/ 2010/06/ exercising-our-remote-application.html

Google. (2011b). *Platform Versions*. Retrieved February 18, 2011, from http://developer.android.com/ resources/ dashboard/platform-versions.html

Google. (2011c). *Security and Permissions*. Retrieved February 18, 2011, from http://developer.android.com/ guide/topics/security/ security.html

GSMA. (2011). *Mobile Privacy*. Retrieved February 18, 2011, from http://www.gsmworld.com/our-work/ public-policy/ mobile_privacy.htm

Hogben, G., Dekker, M. (2011). Smartphone security: Information security risks, opportunities and recommendations for users. Retrieved from:

http://www.enisa. europa.eu/act/it/oar/ smartphones-information-security-risks-opportunities-and-recommendations-for-users/at_download/fullReport

Hypponen, M. (2006). Malware goes mobile. *Scientific American*, 295(5), 70-77.

Kephart, J., White, S. (1991) Directed graph epidemiological models of computer viruses. *1991 IEEE Symposium on Research in Security and Privacy*, 343-359.

Lineberry, A., Richardson, D., Wyatt, T. (2010). *These aren't the permissions you 're looking for*. Retrieved from https://www.defcon.org/images/defcon-18/dc-18-presentations/Lineberry/DEFCON-18-Lineberry-Not-The-Permissions-You-Are-Looking-For.pdf

McAfee. (2010). *2011 Threats predictions*. Retrieved from http://www.mcafee.com/us/resources/reports/rp-threat-predictions-2011.pdf

Microsoft. (2010a). *Security Policy Settings*. Retrieved February 18, 2011, from http://msdn.microsoft.com /en-us/library/bb416355.aspx

Microsoft. (2010b). *Windows Mobile Code Signing*. Retrieved February 18, 2011, from http://msdn.microsoft.com/en-us/windowsmobile/ dd569132.aspx

Microsoft. (2010c). *Windows Mobile Device Security Model*. Retrieved February 18, 2011, from http:// msdn.microsoft.com/en-us/library/ bb416353.aspx

Microsoft. (2010d). *Windows Phone 7 Series Developer General FAQ*. Retrieved February 18, 2011, from http://social.msdn.microsoft.com/Forums/en/windows phone7series/thread/2892a6f0-ab26-48d6-b63c-e38f62eda3b3

Nokia. (2011a). *Capabilities*. Retrieved February 18, 2011, from http://wiki.forum.nokia.com/index.php/ Capabilities

Nokia. (2011b). *Developer certificate*. Retrieved February 18, 2011, from http://wiki.forum.nokia.com/index.php/ Developer_certificate

Nokia. (2011c). *Symbian Signed*. Retrieved February 18, 2011, from https://www.symbiansigned.com/app/page

PAMPAS. (2011). *Pioneering Advanced Mobile Privacy and Security*. Retrieved February 18, 2011, from http://www.pampas.eu.org/

RIM. (2011d). *Java Code Signing Keys*. Retrieved February 18, 2011, from http://us.blackberry.com/ developers/javaappdev/codekeys.jsp

RIM. (2011e). *Security Overview*. Retrieved February 18, 2011, from http://docs.blackberry.com/en/developers/ deliverables/21091/Security_overview_1304155_11.jsp

Seriot, N. (2010). iPhone Privacy. Retrieved from http:// seriot.ch/resources/talks_papers/iPhonePrivacy.pdf

Sindhu, U., Balaouras, S., Hayes, N., Coit, L. (2010). *The Security of B2B: Enabling An Unbounded Enterprise*. Retrieved February 18, 2011, from

http://www.forrester.com/rb/Research/security_of_b2b_enabling_unbounded_enterprise/q/id/56670/t/2

Smartphone Experts (2011). *CrackBerry.com – The #1 Site for BlackBerry Users & Abusers*. Retrieved February 18, 2011, from http://crackberry.com/

Sophos (2011). *Pirated Mac App Store apps pose major risk*. Retrieved February 18, 2011, from http://nakedsecurity.sophos.com/2011/01/07/app-store - developers-leave-purchased-apps-vulnerable-to-piracy/

Symbian Freak. (2011). *S60 3rd ed. FP1 Hacked!* Retrieved February 18, 2011, from http://www. symbian- freak.com/news/008/03/s60_3rd_ed_has_ been_hacked.htm

Theoharidou, M., Xidara, D., Gritzalis, D. (2008). A Common Body of Knowledge for Information Security and Critical Information and Communication Infrastructure Protection. *International Journal of Critical Infrastructure Protection*, 1(1), 81-96.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265 (3), 94-104.