Secure Software Installation on Smartphones*

David Barrera, P.C. van Oorschot
Carleton University

1 Introduction

Smartphones — mobile phones with advanced features such as always-on Internet connectivity, full-featured web browsers and multimedia capabilities — have become extremely popular. Smartphone manufacturers and mobile operating system (OS) vendors are reporting record number of units sold, and thousands of developers are forming communities around each of the popular smartphone platforms. Recent trend analysis anticipates that by 2015, more users will be accessing the web through smartphones than through desktop systems [2]. It isn't entirely surprising to see this shift taking place. Smartphones today are more powerful than desktop computers were 10 years ago; they are more portable, have fast CPUs, large amounts of RAM, and high-speed Internet connectivity. These desktop-like characteristics, coupled with endless innovation from developers, make smartphones a promising platform for the future.

Smartphone OSs ship with applications that provide core phone functionality to the device (e.g., applications to send and receive text messages, make phone calls, etc.). Additional applications such as games, productivity and communications applications are typically written by third party developers. Third party application development has become a key factor in determining a platform's commercial success. This has led major smartphone OS vendors to provide open development tools such as a set of application programming interfaces (APIs), emulators and tools to build applications, even in those cases where the platform itself is not fully open.

There are currently thousands of applications (*apps*) available for the top smartphone platforms which can usually be installed through an on-device store with a few key presses. Allowing such a large number of applications from a variety of developers to be installed with such ease raises security issues. *Can this app be trusted? Can this developer be trusted? Will this app break other apps I have installed?*

This article summarizes how popular smartphone OSs handle the installation of third party applications. We also provide an overview of the common security features in smartphones related to application installation and isolation, present a generalized classification of software installation approaches, and discuss security implications.

1.1 Security Considerations for Smartphones

Mobile phones have traditionally been simple devices capable of performing only basic phone functions. With the release of newer smartphone OSs, mobile phones have started to include advanced desktop-like features, which has caused users (and forced app developers) to think differently about these devices. It is unclear whether users think of their smartphones as computers, since typical computer activities such as installing and updating software are present (albeit simpler), but other activities such as running anti-virus or a firewall on a smartphone are currently uncommon.

Due to their extensive feature-sets, smartphones tend to store more personal data (e.g., pictures, messages, detailed contact information) than their *Plain Old Cellphone* (POC) precursors, making privacy and data leak risks more serious in the smartphone world. Furthermore, always-on connectivity and cloud synchronization facilitates the propagation of locally corrupted data to other synchronization end-points. Blackberry's Enterprise Server (BES[1]) and Android's contacts applications are good examples of where syncing results in contacts and email being stored on remote servers and thus offering additional attack points. Malware infecting the phone

[1] `http://na.blackberry.com/eng/services/business/server/full/`

Smartphone OS	Global Market Share [5]
iPhone	15.7%
Android	22.7%
Blackberry	16%
Symbian	37.6%

Table 1: Year-End 2010 smart phone platform global market share.

could propagate to the cloud and in turn modify contacts on other cloud-connected hosts.

Many smartphones also include GPS receivers to help users get directions and find nearby attractions. Malicious applications can potentially use location information to track or spy on users, leading to serious privacy concerns.

The issues mentioned above are by no means exhaustive, but do provide a flavor of the types of security concerns that can arise from the increasing use of third-party applications. Shabtai et al. [6] provide a comprehensive list of smartphone threats (and their applicability to Android devices).

2 Current Smartphone Platforms

This section reviews the most popular smartphone platforms as of 2010. It includes a brief history, the programming language used in third-party application development and unique features. The platforms discussed account for approximately 94% of the global smartphone market (see Table 1).

2.1 iPhone

iPhone OS (renamed to iOS in July 2010) is Apple's Mac OS based operating system for their line of mobile devices. The iPhone, iPod Touch and iPad run iOS, allowing developers to easily write applications that run on all supported devices. iOS applications are written in Objective-C and are capable of communicating with hardware through a set of published APIs. iOS offers several layers of abstraction to easily create on-screen menus that interact with the user, 2D and 3D graphics, location services and core OS functionality such as threads and network sockets. Application separation and isolation on iOS is achieved through a sandboxing mechanism similar to that of Mac OS X, in which a policy file restricts access to certain device features and data [3]. By default no third-party application can read or write data outside its own directory, which include system files, resources and the kernel. Restricting applications this way requires developers to use registered APIs to access protected resources.

Developers wishing to publish iOS applications must submit them to Apple for approval. While Apple has not published detailed information regarding the criteria underlying its approval process [1], it is generally believed that the company performs a combination of automated and manual verification of submitted apps. If the application is categorized as suitable for public distribution, it is digitally signed by Apple and released to Apple's software clearinghouse, the iTunes App Store. Apple rejects applications that it finds violate intellectual property or go against developer terms of service. Developers have reported cryptic and seemingly subjective rejections for some applications, supporting the consensus that there is at least some manual verification of submitted apps.

2.2 Android

The Open Handset Alliance's Android platform (mainly backed by Google) is an open source Linux-based middleware that runs on top of a Linux kernel. Android powers a variety of devices (over 60 smartphones models, tablets and netbooks as of July 2010) produced by a large number of manufacturers. Hardware support is provided by Linux, while Android provides a device-independent API and user interface. Since the announcement and first release of Android in October 2008, the code base has seen very rapid development, with 3 major releases in 2009 alone.

Applications for Android are written in Java and run in a custom virtual machine called *Dalvik*. Process and file system isolation is primarily provided by making each application run as its own user (standard UNIX UID). By default, applications only have read and write access to files in their own directory. While Dalvik provides some isolation as well, Android makes no security claims or assumptions that the VM itself provides security. This is because application developers can create and invoke libraries written in C/C++, which are run natively, beyond VM boundaries.

A feature that makes Android unique in the smartphone space is that the OS allows applications to interact and use system resources based on a list of permissions labels. The permission-based architec-

ture requires developers to declare any special functionality their apps might need (camera, GPS, access to messages or contact data, etc.). Application developers can specify (in a manifest file) permission labels which *protect* their own interfaces, or labels to *request* access to another application's protected interfaces. Inter-process communication (IPC) is allowed if the callee application has made available unrestricted access to its APIs, or if the calling application has defined necessary permissions in its manifest to access remote APIs. Enck et al. [4] discuss Android's security model including IPC and permission architecture.

Applications for Android can be downloaded through the "Android Market" (Google's controlled app market), or obtained directly through a developer's site or third party app market (also known as *sideloading*). Google has minimal involvement when applications are uploaded to the Android Market and no involvement when applications are distributed from a third party developer site. Google only removes applications from their Market when content is found to violate terms of use or upon confirmation of reported malicious activity. A major distinction from Apple is that Android developers do not have to wait for external approval before their apps become generally available, and banned (removed) Android applications can still be distributed outside the Market.[2]

2.3 Blackberry

Blackberry OS was developed by Research in Motion (RIM). The OS runs on a large number of Blackberry models, and has historically been heavily targeted towards enterprise customers by including features such as push email and groupware support (Microsoft Exchange, Lotus, Novell GroupWise, and BES support).

Blackberry OS supports third party developed applications written in Java. The OS uses sandboxing for isolating applications at runtime, achieved through the Java Virtual Machine (JVM). Developers traditionally wrote Java applications for Blackberry and distributed them through web sites without RIM approval. This changed with the introduction of Blackberry AppWorld (April 2009), where users of newer Blackberry models can access a repository of RIM-approved applications through an on-device application. Even though RIM must approve each submitted application for inclusion in AppWorld, developers are free to host their applications on other servers. Unlike Apple's approval model, having an app approved by RIM is only beneficial for distribution purposes, as apps that are not approved can still be distributed outside the market.

A Blackberry OS feature designed for the enterprise market is the fine-grained control that a company can enable on the devices it hands out to employees. Policies can be "pushed" to Blackberry devices allowing administrators to restrict the functionality that is available to the end-user. For example, policy administrators may decide that applications downloaded from third party web sites are not allowed, but those installed through AppWorld are.

2.4 Symbian

Nokia's Symbian is the world's most widely used smartphone OS, with a smartphone market share of 44% worldwide [5]. The operating system has existed since the early 1990s and is now deployed on hundreds of smartphone models. Symbian used to be a proprietary platform, but was open-sourced by Nokia under the *Symbian^3* branding in February 2010. The OS was designed with integrity, security and low resources in mind (in contrast to the gigahertz chips seen on newer smartphones). While the Symbian platform has been targeted by malware in the past, most attacks have relied on social engineering or direct manipulation of users (e.g., the Cabir[3] worm where users must click "yes" to allow a malicious program to run, and are prompted repeatedly until they do) as opposed to the exploitation of software flaws.

Symbian mandates that all applications be digitally signed, but not all signatures have to be issued by the Symbian Foundation. Developers can self-sign their applications, allowing them to access "user capabilities", which include making phone calls, initiating network connections, and accessing device location data. Applications that need to modify system settings or access core OS files (also known as "system capabilities") must be submitted to the Symbian Signed[4] program for approval. Users can configure Symbian phones to check an online server for the validity of a certificate. While unsigned applications may have limited access to advanced functionality,

[2]Some vendors and carriers customize Android to disable the ability to sideload apps. If sideloading is allowed, however, apps are still subject to the standard Android OS permission model and isolation features.

[3]`http://www.f-secure.com/v-descs/cabir.shtml`

[4]`http://www.symbiansigned.com`

they can still behave maliciously and cause denial of service by executing code repeatedly to drain the battery, or even leak private information. Some carriers disable non-Symbian signed certificates entirely, allowing only signed applications to run on devices controlled by those carriers.

3 Common Security Features

Process and File System Isolation. All smartphone platforms discussed in this article were designed to include some form of application isolation to help protect applications from each other. Isolation refers to the separation of processes and file system access so that each application can run within its own context while remaining unaffected by other (including potentially malicious) applications. On Blackberry and Android, process isolation is provided (at least partially in the case of Android) by the virtual machine (JavaVM and Dalvik, respectively). When running native applications (i.e., not interpreted by a VM), iOS and Symbian provide process isolation at the system level. File system access on smartphones is typically different than on desktop systems. Applications can only read and write data in their own context. POSIX file permissions limit access to files on Android, which uses traditional *read, write* and *execute* bits as well as user and group identifiers (UIDs and GIDs). iOS enforces similar restrictions through sandbox policy files. Some smartphones include a memory card slot, which generally takes a FAT32 formatted card. Since FAT32 does not have file access control (i.e., the `rwx` bits), applications that can read or write to the card have access to all contents, not just data in the application's context.

Application or Code Signing. Code signing involves an authority (in this case the developer of an application or the operating system vendor) digitally signing an application so that at a later time, signature verification can validate that the application was not tampered with and that it originates from the intended author (via private/public keys). While code signing has been adopted by all four smartphone platforms, each platform uses this feature slightly differently to accomplish different goals. Android applications must be self-signed (i.e., developers generate their own keys and sign their applications without Google's involvement) to support verification that subsequent updates of the apps were written by the same developer or organization. This is used for continuity of software updates, but not for initial installs (i.e., there is no signature to compare against on initial installs) and also supports some forms of IPC where the developer wishes to restrict access to calls only to applications signed with the same private key.

On iOS, applications that aren't digitally signed by Apple cannot run on the device. This policy is enforced by the operating system. Apple digitally signs all applications that are approved by their vetting process, giving Apple the final say as to which applications can be distributed on to iOS devices.

The Symbian foundation (through the Symbian Signed program) also performs code signing of applications after a vetting process, but the platform allows the user to configure whether unsigned applications should be allowed always, never, or whether to prompt the user for permission each time. Since end-users have control over what type of code to accept, Symbian does not ultimately have control. Additionally, applications can be self-signed if they require fewer privileges.

Blackberry uses code signing for tracking use of their *restricted* APIs. Certain APIs on the Blackberry OS are restricted, meaning that only signed binaries can make use of the features provided by those APIs. When developers need to use restricted APIs, they purchase a signing key from RIM. Unique keys are generated for each developer, allowing the company to tie billing information to developers and applications, and, if necessary, update certificate revocation lists (checked periodically by devices) to block certain applications from running.

ROM, Firmware and Factory Restore. Today's smartphones may have many of the features of a desktop system. However, one of the key differences is the notion of a *firmware*. The smartphone platforms reviewed in this article have a read-only portion of memory (ROM) that holds the operating system (i.e., the firmware). This area of memory cannot be modified by user-space applications, leading to a much more resilient architecture that is less susceptible to corruption of core system files and libraries. Vendors typically distribute software updates in the form of a firmware that is *flashed* on to the ROM. Most vendors distribute firmware as downloadable files that can be sent to the phone through a USB connection and specialized software. Android handsets can receive and apply *over-the-air* updates while other platforms require a desktop computer to transfer the updated OS onto the smartphone. Users can also restore their smartphones to factory settings by deleting all user stored preferences and reading all configuration set-

tings from ROM.

Kill switches. Remote application revocation and uninstallation, also known as a *kill switch*, is a powerful feature in many smartphone OSs. Kill switches allow the manufacturer to remotely (potentially without user interaction or approval) uninstall or disable an application on a user's smartphone. Typically working closely with the platform's application store or market, kill switches offer a mechanism to control the spread of malicious applications, but can also facilitate excessive control (or what might be viewed by some as censorship or anti-competitive practices) by vendors. iOS and Android have kill switches that work in conjunction with their application stores. It is unclear whether Blackberry and Symbian have a similar system in place.

4 Classification of Software Installation Models on Smartphones

To better understand and compare different systems, we find it useful to reference a common framework. To that end, we present a classification of software installation models for third party applications on smartphones. When smartphones are shipped, their initial factory configuration generally includes basic phone software and core applications. The classification below applies to software (i.e., third party software) beyond core vendor-provided apps.

We see three generic models for software installation, classified by the level of control the smartphone OS or hardware vendor has over software installation and management. The *walled garden* model provides the vendor with the most control. In the *end-user control* model, the vendor has practically no control over software once the phone is sold. The *guardian* model is a middle ground that can be adapted to several environments. We argue that no smartphone OS falls under a single category, and each category has advantages and disadvantages.

4.1 Walled Garden Model

The walled garden model gives the smartphone vendor full control over third party installation of software on end-user devices. Users can only install software that has been approved and made available through a vendor's app market or clearinghouse. Applications can be removed from the clearinghouse by the vendor, as well as remotely uninstalled or disabled on user's devices (see *Kill switches* in Section 3). Code signing is an essential part of this model, since it provides a reliable technical mechanism to prove that an app was accepted by the vendor and has not been modified. The walled garden model leaves most of the security decisions and testing up to the vendor, giving even non-technical users a (perhaps unfounded) worry-free experience of smartphones.

While this model is subject to controversy due to the vendor's totalitarian control over the user experience on the device, it provides a strong set of tools to control platform security. In the event that a malicious application is detected (even post-vetting), it can be uninstalled from all devices. The vetting process itself, in conjunction with the single point of software entry on to the device, also allows the vendor to monitor trends and tightly control use of features on the device.

4.2 Guardian Model

In the guardian model, security decisions are delegated to a knowledgeable third party (i.e., the *guardian*). The guardian role can be assigned to a variety of entities ranging from the OS vendor (in which case the guardian model becomes more similar to the walled garden model), the mobile phone carrier, an acknowledged expert acting on behalf of a less knowledgeable group of users, or an enterprise system administrator who already controls policy on other devices. The guardian is typically in charge of making most of the fundamental security decisions (e.g., which apps are allowed to be installed, what services they are allowed to access on the device); end users are minimally involved with making decisions thereafter. The guardian may also perform a less rigorous application vetting process (e.g., banning applications that violate corporate policy). This method provides a flexible middle ground for software installation that can be fine-tuned according to the required level of security. If the guardian role is assigned to the end-user, this model moves closer to the end-user control model.

4.3 End-user control Model

In this model, the user is responsible for all software installation and software security decisions. Users are free to install software from any source (website, memory card, application marketplace), understanding the risk that any or all applications could be ma-

licious since there is no application vetting. The end-user control model should ideally enforce any available strong operating system security features such as application isolation to limit the negative impact of malicious applications on user experience. This is a difficult balance to reach, since users may be required to answer puzzling questions (e.g., questions which users do not have the technical expertise or detailed knowledge to answer) about software either at install time, or at resource access time (e.g., "Do you want to allow application A to read phone state?"). Third party applications are distributed to end users with minimal involvement from the phone vendor or carrier, reducing overhead costs.

4.4 Classifying Existing Systems

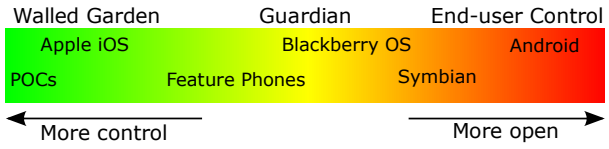

Figure 1: Approximate binning of smartphone platforms across the three generic software installation models. Plain old cellphones (POCs) and feature phones are listed for reference.

The software installation models presented are fairly generic, and as such, mobile platforms will not fit perfectly into any single one. Figure 1 shows an approximate binning of platforms into the different models.

Apple iOS falls mostly into the walled garden model, since Apple ultimately has decisive power over what applications are made available on their App Store. However, iOS isn't entirely a walled garden OS; for example in some instances the user is prompted to make security decisions (e.g., allowing access to geolocation data). The OS itself is also preloaded with policy files, resembling a (quite restrictive) guardian model.

Android is at the other end of the spectrum, fitting tightly into the end-user control model and relying heavily on users to keep their devices clear of malware. Some carriers might choose to ship a branded version of Android which is customized to their specific needs. In these cases, Android moves more towards the guardian model (where the guardian is the carrier). Of course, since the Android OS is open source, customization can result in variations yet to be seen. Finally, Google can (and has) made use of the remote kill switch,[5] showing some resemblance to the properties of the Walled Garden.

Blackberry OS most closely resembles the guardian model. Depending on the environment, the guardian may play an important role in configuring policy for Blackberry devices (typical corporate use). The guardian could be the carrier as well, configuring the device for more flexible use and involving the user only under certain conditions.

Symbian falls somewhere between guardian and end-user control, but it is more difficult to locate on the continuum of Figure 1. Many of the OS security features are configured by Symbian, but some (such as bypassing unsigned application warnings) are user-configurable.

We list plain old cellphones (POCs) as the canonical example of the walled garden model, since manufacturers and carriers do not typically allow or support any phone modification (including app installation) post-sale. We also place feature phones between the guardian and walled garden models, since these devices allow app installation, but carriers will often act as guardians disabling features and services on the device as they see fit.

5 Controlled markets for third party applications

A trend in the smartphone space has been for each smartphone vendor to provide a third party application repository (i.e., a controlled market) that acts as a central location for application vetting, application (sales and) distribution, or both. Depending on the software installation model used, the market may have a rigorous vetting process in order to get apps included. Other markets might be used for end-user convenience only, providing an on-device location for searching, rating and buying additional applications.

There is an important difference between the vetting process for app inclusion in a controlled market and the way 'approved status' is denoted. Having an app appear in a market (as is the case in the Android Market) does not imply that the app has undergone substantial vetting. Approval status can be denoted by a digital signature, verified by a corresponding public key on the device; or allowing placement in a closed market, verified by validating the software source (e.g., through SSL or other types of endpoint

[5] http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html

verification); or alternatively, both methods can be used in parallel.

5.1 Controlled markets under each model

Depending on the software installation model, controlled markets may be used for different purposes. In each of the previously defined models, the market acts as a repository which allows an on-device application to conveniently search for, install and rate applications. Controlled markets also help developers reach a large number of users by allowing the upload of their application to a central location, reaching customers in many countries, carriers, and devices.

In the walled garden model, the application market has a secondary purpose: it acts as a choke point for allowing or rejecting applications, giving the OS vendor full control over what applications are available to end-users. The added control comes at the cost of scalability and thoroughness in testing.

The end-user control model uses a controlled market only for distribution purposes. Thus, third-party markets such as those provided by carriers are possible under this model. The OS vendor will rarely be involved in app testing or vetting, allowing developers and users to interact more freely. Due to a minimal involvement policy, end-user model app markets might rely more on crowd-sourced vetting or recommendation systems.

The guardian model uses the market for some control, but has a heavier focus on behaving as an application distribution platform. The app market allows download, but the installation is still controlled by configurable policy on the phone itself. While there could be an application approval process, developers can still (policy permitting) distribute applications outside the controlled market (e.g., through their own website). Controlled markets on platforms using the guardian model may serve as a repository for "premium" applications that have undergone some form of testing (some of which are described in Section 5.2).

5.2 Application Vetting Tests

This section describes some of the reported tests that vendors run on applications during the app vetting process. This includes information obtained informally and from anecdotal reports; most smartphone OS vendors do not publicly explain their testing process. Of the platforms considered herein, only Symbian publishes a list of tests performed,[6] and only compiled binaries (i.e., not source code files) are reviewed by vendors.

Smoke tests. These tests usually involve a quick overview inspection of the application to ensure that it does not catastrophically fail. Generally this is not a thorough test, but rather an initial sanity check to verify that the application is worth the full testing process (provided there is one). Smoke tests help filter out broken applications submitted by mistake, as well as poorly written applications. This type of test must be simple to perform in an automated fashion, reducing costs albeit sometimes at the expense of accuracy. It is our understanding that all controlled markets perform at least basic smoke testing on submitted applications.

Hidden API checks. Mobile operating systems, like their desktop counterparts, contain APIs that are reserved for system applications. These APIs are generally hidden from developer documentation, and intended to be used only by the OS vendor. Developers sometimes use hidden APIs (by either guessing function names in a common namespace or reverse engineering the OS) to obtain direct access to low-level functionality or to speed up their application by avoiding unnecessary layers of abstraction (especially in graphics and media code). Developer use of hidden APIs is regarded as a poor programming practice, since these APIs could change with future OS releases. Static code analysis and debugging can help identify use of hidden APIs, but all instances might not be revealed. Manual testing and fuzzing may be required for this test.

Functionality checks. These tests verify that the application being submitted is capable of undergoing "typical expected use" without interfering with other installed applications. Details of how such tests are performed are not always made available by smartphone vendors, but we expect manual testing is required. Functionality tests involve simulated real word application use to ensure the application opens, closes, does not crash, etc. Other checks may include verifying that an application does not disrupt basic phone functionality (e.g., the ability to receive a phone call or message) or drain the battery. Some vendors might also perform "second-stage" testing on the most popular applications in their application repository.

Intellectual property, liability, and TOS

[6] `https://www.symbiansigned.com/app/page/overview/testcriteria`

checks. These checks involve verifying that the submitted application does not violate Terms of Service (established by the OS vendor or carrier) or infringe on intellectual property. Tests in this category are usually performed to limit the vendor's liability in the event of a legal dispute surrounding the application once it has been approved. Such checks can be partially automated by looking for specific trademarked keywords or files, but likely require some manual inspection if searching for objectionable content or simply rely on independent notification or complaints by third parties.

User interface checks. Some vendors place heavy emphasis on the user interface of the application in an attempt to deliver a more consistent user experience. For these vendors, testing the user interface (i.e., the placement of buttons, color schemes, navigation within the application, etc.) is important. Failure to comply with established UI guidelines could result in the application being rejected from a vendor's controlled market. Checks in this category are believed to be done manually rather than in an automated way.

Bandwidth checks. Using excessive amounts of bandwidth can severely impact a network. Applications that stream Internet radio or download large files may be further tested to see if they operate within a network operator's infrastructure constraints.

Security checks. Until concrete evidence becomes available, the most prudent course is to assume that no security-specific tests are performed by any vendors during the vetting stage.

6 Conclusions

We have presented an overview of how software installation is handled by four of the most popular smartphone operating systems. To help differentiate the systems, we have presented a classification for the different software installation models on smartphones, each offering advantages and drawbacks in the smartphone marketplace. Depending on the environment where smartphones are deployed, one model may be more appropriate than another, but there is no perfect one-size-fits-all approach. Factors to consider when selecting a smartphone platform or software installation model include the type of environment the phone will be used in (e.g., corporate, personal), the level of expertise users have, and their usage patterns (e.g., novice, expert, frequent users).

Smartphones are currently not high targets for malicious software, but as their popularity continues to increase (including as both payment devices and for access to sensitive information), the value of exploiting smartphones increases. Secure software installation and control mechanisms will play a key role in helping smartphones avoid replication of many security issues that currently plague desktop systems.

References

[1] Apple Answers the FCCs Questions. `http://www.apple.com/hotnews/apple-answers-fcc-questions/` Retrieved Nov 2, 2010.

[2] Morgan Stanley Internet Trends. `http://www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf` Retrieved May 10, 2010.

[3] J. Anderson, J. Bonneau, and F. Stajano. Inglorious Installers: Security in the Application Marketplace. In *Proceedings of the 9th Workshop on the Economics of Information Security*, 2010.

[4] W. Enck, M. Ongtang, and P. McDaniel. Understanding Android Security. *IEEE Security & Privacy*, 7(1):50–57, 2009.

[5] Gartner Inc. Gartner says worldwide mobile device sales to end users reached 1.6 billion units in 2010; Smartphone sales grew 72 percent in 2010. `http://www.garnter.com/it/page.jsp?id=1543014` Retrieved Dec. 1, 2010.

[6] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google Android: A Comprehensive Security Assessment. *IEEE Security & Privacy*, 8(2):35 –44, Mar. 2010.