Secure Software Installation on Smartphones

This overview of iOS, Android, BlackBerry, and Symbian security frameworks includes a novel classification of third-party-application installation models. It also discusses how controlled app marketplaces fit in the smartphone security ecosystem.

David Barrera and Paul van Oorschot
Carleton University

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-OS vendors are selling record numbers of units, and thousands of developers are forming communities around each of the popular smartphone platforms. A recent analysis suggests that by 2015, more users will access the Web through smartphones than through desktop systems.[1] This shift isn't entirely surprising. Smartphones are more powerful today than desktop computers were 10 years ago; they are more portable and 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 providing core functionalities, such as making phone calls and sending and receiving text messages. Typically, additional apps such as games and productivity and communications apps come from third-party developers. Third-party-app 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 APIs, emulators, and tools to build apps, even when the platform itself isn't fully open.

Currently, the top smartphone platforms have thousands of available apps, which users can usually install through an on-device store with a few key presses. Letting users install so many apps from a variety of developers with such ease raises security issues. Users must consider whether they can trust an app and its developer and whether the app will break any other apps they've installed.

Here, we describe how the four most popular smartphone OSs—Apple's iOS, Android, BlackBerry, and Symbian—handle installation of third-party apps, focusing on security. Our research on this topic led us to a generalized classification of application installation methods, which have important implications for smartphone security.

Security Considerations for Smartphones

Mobile phones were once simple devices capable of performing only basic phone functions. With the release of newer smartphone OSs, mobile phones began to include advanced desktop-like features, which has caused users (and forced app developers) to think differently about these devices. Whether users think of their smartphones as computers is unclear. Typical computer activities such as installing and updating software are present—albeit simpler—but other activities such as running antivirus or firewalls are currently uncommon.

Because of extensive feature sets, smartphones tend to store more personal data (for example, pictures, messages, and detailed contact information)

 1540-7993/11/$26.00 © 2011 IEEE

than their plain-old cell phone (POC) precursors. So, privacy and data-leak risks in the smartphone world are more serious. Furthermore, always-on connectivity and cloud synchronization facilitate the propagation of locally corrupted data to other synchronization end points. For example, in BlackBerry's Enterprise Server (BES) and Android's contacts apps, syncing results in contacts and email being stored on remote servers and thus offering additional attack points. Malware infecting the phone 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 apps can potentially use location information to track or spy on users, leading to serious privacy concerns.

These issues are by no means exhaustive but provide a flavor of the types of security concerns that can arise from increasing third-party-app use. Asaf Shabtai and his colleagues provide a comprehensive list of smartphone threats.[2]

Current Smartphone Platforms

As of December 2010, iOS, Android, BlackBerry, and Symbian accounted for approximately 92 percent of the global smartphone market (measured by total units sold in 2010; see Table 1).

iOS

Apple's iOS (originally called the iPhone OS) is based on the Mac OS. The iPhone, iPod Touch, and iPad all run it, letting developers easily write apps that run on all those devices. iOS apps are written in Objective-C and can communicate with hardware through a set of published APIs. iOS offers several abstraction layers to easily create onscreen interactive menus, 2D and 3D graphics, location services, and core OS functionality such as threads and network sockets.

iOS achieves application separation and isolation through a sandbox mechanism similar to that of Mac OS X, in which a policy file restricts access to certain device features and data.[4] By default, no third-party app can read or write data outside its own directory, which includes system files, resources, and the kernel. Restricting apps this way requires developers to use registered APIs to access protected resources.

Developers wishing to publish iOS apps must submit them to Apple for approval. Apple hasn't published detailed information regarding the criteria underlying its approval process.[5] It's generally believed that the company employs both automated and manual verification of submitted apps. If an app is categorized as suitable for public distribution, Apple digitally signs it and releases it to Apple's software clearinghouse, the iTunes App Store. Apple rejects apps that it finds violate intellectual property laws or developer terms of service. Developers have reported cryptic and seemingly subjective rejections of some apps, supporting the consensus that Apple performs at least some manual verification.

Table 1. Year-end 2010 smartphone platform global market share.[3]

Smartphone OS	Global market share (%)
iOS	15.7
Android	22.7
BlackBerry	16.0
Symbian	37.6

Android

The Open Handset Alliance's Android platform (mainly backed by Google) is open source Linux-based middleware that runs on top of a Linux kernel. Android powers a variety of smartphones, tablets, and netbooks from many manufacturers. Linux provides hardware support, and Android provides a device-independent API and UI. Since Android's announcement and first release in October 2008, the code base has seen rapid development, with three major releases in 2009 alone.

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

One unique Android feature is that it lets apps interact and use system resources on the basis of a list of permission labels. Developers must declare any special functionality their apps might need, such as a camera, GPS, and access to messages or contact data. They can specify (in a manifest file) permission labels that protect their own interfaces, or labels to request access to another app's protected interfaces. Interprocess communication (IPC) is allowed if the callee app allows unrestricted access to its APIs or if the calling app has defined the necessary permissions in its manifest to access remote APIs. William Enck and his colleagues discuss Android's security model, including IPC and the permission architecture.[6]

Android apps can be downloaded through the Android Market—Google's controlled app market-

place—or obtained directly through a developer's site or third-party-app marketplace, also called *sideloading*. Google has minimal involvement when apps are uploaded to the Android Market and no involvement when apps are distributed from a third-party developer site. Google removes apps from the Android Market when their content violates terms of use or when reported malicious activity is confirmed. A major difference from iOS is that Android developers don't have to wait for external approval before their apps become generally available and can still distribute banned Android apps outside the Android Market.

BlackBerry

Research in Motion (RIM) developed the BlackBerry OS. It runs on many BlackBerry models and has historically targeted enterprise customers by including features such as push email and groupware support (for example, Microsoft Exchange, Lotus, Novell GroupWise, and BES support).

BlackBerry supports third-party apps written in Java. It uses sandboxing to isolate apps at runtime, through the Java Virtual Machine (JVM). Developers traditionally wrote Java apps for BlackBerry and distributed them through websites without requiring RIM approval. This changed in April 2009 with the introduction of BlackBerry App World, in which users of newer BlackBerry models can access a repository of RIM-approved apps through an on-device app. Even though RIM must approve each submitted app for inclusion in App World, developers can host their apps on other servers. Unlike Apple's approval model, having a RIM-approved app is only beneficial for distribution purposes; unapproved apps can still be distributed outside the market.

BlackBerry OS gives companies fine-grained control of devices they distribute to employees. Administrators can push policies to BlackBerry devices, letting them restrict the functionality available to users. For example, policy administrators might decide that apps downloaded from third-party websites aren't allowed but that those installed through App World are.

Symbian

Nokia's Symbian is the most widely used smartphone OS. It has existed since the early 1990s and is now deployed on hundreds of smartphone models. Symbian was a proprietary platform until February 2010, when Nokia open-sourced it under the Symbian^3 branding. Nokia designed the OS with integrity, security, and low resources in mind, in contrast to the gigahertz chips on newer smartphones. Although malware has targeted Symbian in the past, few attacks exploited software flaws. Rather, they relied on social engineering or direct user manipulation—for example, the Cabir worm repeatedly prompts users to click "yes" to allow a malicious program to run.

Symbian mandates that all apps be digitally signed, but not all signatures have to be issued by the Symbian Foundation. Developers can self-sign their apps, letting them access "user capabilities," including making phone calls, initiating network connections, and accessing device location data. Developers must submit apps that modify system settings or access core OS files or system capabilities to the Symbian Signed program for approval. Users can configure Symbian phones to check an online server for a certificate's validity. Unsigned apps might have limited access to advanced functionality. However, 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 apps to run on devices they control.

Common Security Features

The main security features common to all four OSs involve process and file system isolation; app or code signing; ROM, firmware, and factory restore; and kill switches.

Process and File System Isolation

All four OSs include some form of application isolation to help protect apps from each other. Isolation separates processes and file system access so that each app can run in its own context while remaining unaffected by other, potentially malicious, apps. BlackBerry and Android provide process isolation (at least partially in the case of Android) through VMs (JVM and Dalvik, respectively). When running native apps (not interpreted by a VM), iOS and Symbian provide process isolation at the system level.

Typically, file system access on smartphones differs from that on desktop systems. Applications can read and write data only 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. iOS enforces similar restrictions through sandbox policy files. Some smartphones include a memory card slot, which generally takes a FAT32 formatted card. Because FAT32 doesn't have file access control (that is, the read, write, and execute bits), apps that can read or write to the card can access all content, not just data in the app's context.

App or Code Signing

In code signing, an authority such as the app developer or OS vendor digitally signs an app so that signature verification can later validate that the app wasn't tampered with and that it originates from the

intended author (via key pairs). Although all four OSs have adopted code signing, each uses this feature slightly differently to accomplish different goals. Android apps must be self-signed. Without Google's involvement, developers generate their own keys and sign their apps to verify that the same developer or organization wrote any app updates. This mechanism is used for continuity of software updates, but not for initial installs (because there's no signature to compare against on initial installs). It supports some forms of IPC in which developers wish to restrict access to calls to only apps signed with the same private key.

On iOS, apps that Apple hasn't digitally signed can't run on the device. The OS enforces this policy. Because Apple digitally signs all apps that are approved by its vetting process, it has the final say as to which apps are distributed onto iOS devices.

The Symbian Foundation (through the Symbian Signed program) also performs app code signing after a vetting process, but the platform lets the user configure whether unsigned apps should be allowed always or never, or to prompt the user each time. In addition, apps can be self-signed if they require fewer privileges.

BlackBerry uses code signing to track restricted API use. Certain APIs on the BlackBerry OS are restricted, meaning that only signed binaries can use those API features. When developers need to use restricted APIs, they purchase a signing key from RIM. Each developer receives a unique key, letting the company tie billing information to developers and apps and, if necessary, update certificate revocation lists to block certain apps from running.

ROM, Firmware, and Factory Restore

Today's smartphones have many desktop system features, but one key difference involves firmware. The four OSs have a read-only portion of memory that holds the OS (that is, the firmware). User-space apps can't modify this memory, leading to a more resilient architecture that's less susceptible to corruption of core system files and libraries. Vendors typically distribute software updates in the form of firmware flashed onto the ROM. They distribute the 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, whereas other platforms currently 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 settings from ROM.

Kill Switches

Remote app revocation and uninstallation, also called a kill switch, is a powerful feature in many smartphone OSs. Kill switches let the manufacturer remotely (potentially without user interaction or approval) uninstall or disable an app on a user's smartphone. Typically working closely with the platform's app marketplace, kill switches offer a mechanism to control the spread of malicious apps. However, they can also facilitate excessive vendor control and might potentially be abused for censorship or an anticompetitive practice. iOS and Android have kill switches that work in conjunction with their app marketplaces. Whether BlackBerry and Symbian employ a similar system is unclear.

Classifying Software Installation Models

We see three generic software installation models, classified by the level of control the smartphone OS or hardware vendor has over software installation and management: the *walled-garden model*, the *guardian model*, and the *user control model*.

The Walled-Garden Model

This model gives the smartphone vendor full control over third-party software installation on users' devices. Users can install only software that has been approved and made available through a vendor's app marketplace or clearinghouse. The vendor can remove apps from the clearinghouse and can remotely uninstall or disable them on users' devices using a kill switch. Code signing is an essential part of this model because it provides a reliable technical mechanism to prove that an app was accepted by the vendor and hasn't been modified. This model leaves most of the security decisions and testing up to the vendor, giving even nontechnical users a (perhaps unfounded) worry-free smartphone experience.

Although this model is subject to controversy because of the vendor's totalitarian control over the user experience, it provides a strong toolset to control platform security. Upon detecting a malicious app, the vendor can uninstall it from all devices. A vetting process, in conjunction with the single point of software entry onto the device, also lets the vendor monitor trends and tightly control feature use.

The Guardian Model

This model delegates security decisions to a knowledgeable third party. A variety of entities can play this guardian role:

- 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

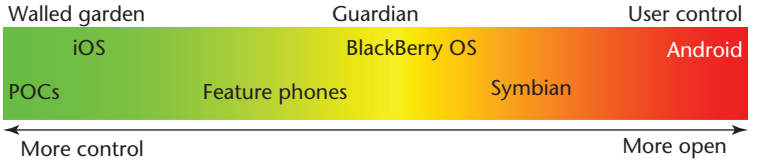

Figure 1. An approximate binning of smartphone platforms across the three generic software installation models. For reference, the figure includes plain-old cell phones (POCs) and feature phones.

- 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 (for example, which apps may be installed and which services they're allowed to access). Thereafter, users are minimally involved with the decisions. The guardian might also perform less rigorous app vetting—for instance, banning apps that violate corporate policy.

This method provides a flexible middle ground for software installation that can be fine-tuned according to the required security level. If the guardian is the user, this model moves closer to the user control model.

The User Control Model

Here, the user is responsible for all software installation and software security decisions. Third-party apps are distributed to users with minimal involvement from the phone vendor or carrier, reducing overhead costs. Users can install software from any source (website, memory card, or app marketplace), understanding the risk that, because there's no app vetting, any or all apps could be malicious.

Ideally, this model should enforce any available strong OS security features, such as application isolation, to limit malicious apps' negative impact on the user experience. This balance is difficult to reach because users might be required to answer puzzling questions about software at either installation time or resource access time. Users might not have the technical expertise or detailed knowledge to answer questions such as, "Do you want to allow app A to read the phone state?"

Classifying Existing Systems

Figure 1 shows an approximate binning of platforms into the different models.

iOS falls mostly into the walled-garden model because Apple ultimately has the power over which apps are available at its App Store. However, iOS isn't entirely a walled-garden OS; for example, in some instances the user must make security decisions, including whether to allow access to geolocation data. The OS itself is also preloaded with policy files, resembling a (relatively restrictive) guardian model.

Android is at the other end of the spectrum, fitting tightly into the 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 customized to their specific needs. In these cases, Android moves more toward the guardian model, in which the carrier is the guardian. Of course, because the Android OS is open source, customization can result in variations yet to be seen. Finally, Google can (and has) employed the remote kill switch, showing some resemblance to the walled-garden model.

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

Symbian falls somewhere between the guardian and user control models but is more difficult to locate on the continuum in Figure 1. Symbian configures many of its security features, but some are user configurable, such as bypassing unsigned-app warnings.

We list POCs as the canonical example of the walled-garden model because manufacturers and carriers don't typically allow or support any phone modification, including app installation, postsale. We place feature phones between the walled-garden and guardian models. These devices allow app installation, but carriers will often act as guardians, disabling features and services on the device as they see fit.

Controlled Marketplaces for Third-Party Applications

A trend has been for each smartphone vendor to provide a third-party-app repository (a controlled marketplace) that acts as a central location for app vetting, app distribution, or both. Depending on the software installation model, the marketplace might require rigorous vetting before including an app. Other marketplaces might be for user convenience only, providing an on-device location for searching for, rating, and buying apps.

An app's presence in a marketplace doesn't imply that it has undergone substantial vetting. The approved status can be denoted in three ways. The first is by a digital signature, verified by a corresponding public key on the device. The second is by the app's placement in a closed market, verified by validating the software's source (for example, through a Secure Sockets Layer or other types of end-point verification). Finally, both methods could be employed in parallel.

Controlled Markets under Each Model

Depending on the software installation model, controlled markets might serve different purposes. In each model, the market is a repository that lets an on-device app conveniently search for, install, and rate apps. Controlled markets also help developers reach a large number of users—in many countries, on various carriers and devices—by letting them upload their apps to a central location. In the walled-garden model, the app marketplace has a secondary purpose: it's a choke point for allowing or rejecting apps, giving the OS vendor full control over which apps are available to users. The added control comes at the cost of scalability and thoroughness in testing, because the vendor must inspect each submitted application.

The user control model uses a controlled market only for distribution. So, it allows third-party markets such as those provided by carriers. The OS vendor will rarely be involved in app testing or vetting, letting developers and users interact more freely. Because of a minimal-involvement policy, user-control-model app marketplaces might rely more on crowd-sourced vetting or recommendation systems.

The guardian model uses the marketplace for some control but focuses more on app distribution. The app marketplace allows downloads, but a configurable policy on the phone controls installation. Although an app approval process could exist, developers can distribute apps outside the controlled market—for example, through their own websites. Controlled markets on platforms using the guardian model might serve as a repository for "premium" apps that have undergone some testing.

Application Vetting Tests

We describe some tests that vendors run on apps during vetting. This includes information we obtained informally as well as anecdotal reports; most smartphone OS vendors don't publicly explain their testing process. Of the four platforms we discuss, only Symbian publishes a list of tests it has performed, and the vendors review only compiled binaries, not source code files.

Smoke tests. These tests usually involve a quick overview inspection of the app to ensure that it doesn't fail catastrophically. Generally, these tests aren't thorough but are an initial sanity check to verify that the app is worth a full testing process, provided one exists. Smoke tests help filter out both broken apps submitted by mistake and poorly written apps. These tests must be simple to perform in an automated fashion, thereby reducing costs, albeit sometimes at the expense of accuracy. Our understanding is that all controlled marketplaces perform at least basic smoke testing.

Hidden-API checks. Mobile OSs, like their desktop counterparts, contain APIs reserved for system apps. 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 directly access low-level functionality or speed up their app by avoiding unnecessary abstraction layers, especially in graphics and media code. This use is considered poor programming practice because the APIs could change with future OS releases. Static code analysis and debugging can help identify hidden APIs but might not reveal all instances. This test might require manual testing and fuzzing.

Functionality checks. These tests verify that the app can undergo typical expected use without interfering with other installed apps. Smartphone vendors don't always release details of how they perform such tests, but we expect manual testing is required. Functionality checks involve simulated real-world app use to ensure the app opens, closes, doesn't crash, and so forth. Other checks might include verifying that an app doesn't drain the battery or disrupt basic phone functionality, such as the ability to receive a phone call. Some vendors might also perform further testing on the most popular apps in their app repository.

Intellectual property, liability, and terms-of-service checks. These checks verify that an app doesn't violate vendor or carrier terms of service or infringe on intellectual property. Vendors usually perform these tests to limit their liability in the event of a legal dispute surrounding the app once it's been approved. Such checks can be partially automated by looking for specific trademarked keywords or files. However, they likely require some manual inspection if they're searching for objectionable content or if they simply rely on independent notification or complaints by third parties.

UI checks. Some vendors heavily emphasize the app's UI in an attempt to deliver a more consistent user experience. For these vendors, testing the UI—including the placement of buttons, color schemes, and navigation—is important. Failure to comply with established UI guidelines could result in the app being rejected from a vendor's controlled market. We believe these checks are manual.

Bandwidth checks. Using excessive bandwidth can severely impact a network. Applications that stream Internet radio or download large files might be further tested to see whether 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 vendors perform security-specific tests during vetting.

Depending on the smartphone environment, one installation model might be more appropriate than another, but there's no perfect one-size-fits-all approach. Factors to consider when selecting a smartphone platform or software installation model include the environment in which the phone will be used (for example, corporate or personal), users' expertise level, and their usage patterns (for example, novice, expert, or frequent users).

Currently, smartphones aren't high targets for malicious software, but as their popularity continues to increase (particularly for making payments and accessing sensitive information), the value of exploiting them increases. Secure software installation and control mechanisms will play a key role in helping smartphones avoid replication of many security issues that plague desktop systems. □

References

1. "Internet Trends," Morgan Stanley, 12 Apr. 2010; www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf.
2. A. Shabtai et al., "Google Android: A Comprehensive Security Assessment," *IEEE Security & Privacy*, vol. 8, no. 2, 2010, pp. 35–44.
3. "Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010," Gartner, 9 Feb. 2011; www.gartner.com/it/page.jsp?id=1543014.
4. J. Anderson, J. Bonneau, and F. Stajano, "Inglorious Installers: Security in the Application Marketplace," *Proc. 9th Workshop Economics of Information Security*, 2010; http://weis2010.econinfosec.org/papers/session3/weis2010_anderson_j.pdf.
5. "Apple Answers the FCC's Questions," Apple, 2011; www.apple.com/hotnews/apple-answers-fcc-questions.
6. W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," *IEEE Security & Privacy*, vol. 7, no. 1, 2009, pp. 50–57.

David Barrera *is a computer science PhD student at Carleton University. His research interests include IPv6, data visualization, and smartphone security. Barrera has an MS in computer science from Carleton University. Contact him at dbarrera@ccsl.carleton.ca.*

Paul van Oorschot *is a computer science professor at Carleton University, where he's the Canada Research Chair in Authentication and Computer Security. Van Oorschot has a PhD in computer science from the University of Waterloo. He was the program chair of Usenix Security 2008 and the program cochair of the 2001 and 2002 Network and Distributed System Security Symposiums, and is a coauthor of the* Handbook of Applied Cryptography *(CRC Press, 2001). Contact him at paulv@scs.carleton.ca.*