# A Window Into Mobile Device Security

## Examining the security approaches employed in Apple's iOS and Google's Android

Carey Nachenberg
VP, Fellow

Contents

Executive Summary ........................................ 1
Introduction ........................................ 1
Mobile Security Goals ........................................ 2
Web-based and network-based attacks ..... 2
Malware ........................................ 2
Social Engineering Attacks ........................................ 3
Resource Abuse ........................................ 3
Data Loss ........................................ 3
Data Integrity Threats ........................................ 3
Device Security Models ........................................ 3
Apple iOS ........................................ 4
Android ........................................ 10
iOS vs. Android: Security Overview ........................................ 17
Device Ecosystems ........................................ 17
Mobile Security Solutions ........................................ 20
Mobile Antivirus ........................................ 20
Secure Browser ........................................ 21
Mobile Device Management (MDM) .......... 21
Enterprise Sandbox ........................................ 21
Data Loss Prevention (DLP) ........................................ 22
Conclusion ........................................ 22

## Executive Summary

The mass-adoption of both consumer and managed mobile devices in the enterprise has increased employee productivity but has also exposed the enterprise to new security risks. The latest mobile platforms were designed with security in mind—both teams of engineers attempted to build security features directly into the operating system to limit attacks from the outset. However, as the paper discusses, while these security provisions raise the bar, they may be insufficient to protect the enterprise assets that regularly find their way onto devices. Finally, complicating the security picture is the fact that virtually all of today's mobile devices operate in an ecosystem, much of it not controlled by the enterprise—they connect and synchronize out-of-the-box with third-party cloud services and computers whose security posture is potentially unknown and outside of the enterprise's control.

## Introduction

With so many consumer devices finding their way into the enterprise, CIOs and CISOs are facing a trial by fire. Every day, more users are using mobile devices to access corporate services, view corporate data, and conduct business. Moreover, many of these devices are not controlled by the administrator, meaning that sensitive enterprise data is not subject to the enterprise's existing compliance, security, and Data Loss Prevention policies.

To complicate matters, today's mobile devices are not islands—they are connected to an entire ecosystem of supporting cloud and PC-based services. Many corporate employees synchronize their device(s) with at least one public cloud based service that is outside of

the administrator's control. Moreover, many users also directly synchronize their mobile device with their home computer to back up key device settings and data. In both scenarios, key enterprise assets may be stored in any number of insecure locations outside the direct governance of the enterprise.

In this paper, we will review the security models of the two most popular mobile platforms in use today, Android and iOS, in order to understand the impact these devices will have as their adoption grows within enterprises.

# Mobile Security Goals

One thing is clear—when it comes to security, the two major mobile platforms share little in common with their traditional desktop and server operating system cousins. While both platforms were built upon existing operating systems (iOS is based on Apple's OSX operating system and Android is based on Linux), they each employ far more elaborate security models that are designed into their core implementations. The ostensible goals of their creators: to make the platforms inherently secure rather than to force users to rely upon third-party security software.

So have Apple and Google been successful in their quest to create secure platforms? To answer this question, we will provide a thorough analysis of each platform's security model and then analyze each implementation to determine its effectiveness against today's major threats, including:

- Web-based and network-based attacks.
- Malware
- Social engineering attacks.
- Resource and service availability abuse.
- Malicious and unintentional data loss.
- Attacks on the integrity of the device's data.

The sections below provide a brief overview of each attack class.

## *Web-based and network-based attacks*

These attacks are typically launched by malicious websites or compromised legitimate websites. The attacking website sends malformed network content to the victim's browser, causing the browser to run malicious logic of the attacker's choosing. Once the browser has been exploited, the malicious logic attempts to install malware on the system or steal confidential data that flows through the Web browser.

A typical Web-based attack works as follows: an unsuspecting user surfs to a malicious Web page. The server on which the page is hosted identifies the client device as running a potentially vulnerable version of the operating system. The attacking website then sends down a specially crafted set of malicious data to the Web browser, causing the Web browser to run malicious instructions from the attacker. Once these instructions have control of the Web browser, they have access to the user's surfing history, logins, credit card numbers, passwords, etc., and may even be able to access other parts of the device (such as its calendar, the contact database, etc.).

## *Malware*

Malware can be broken up into three high-level categories: traditional computer viruses, computer worms, and Trojan horse programs. Traditional computer viruses work by attaching themselves to legitimate host programs much like a parasite attaches itself to a host organism. Computer worms spread from device to device over a network. Trojan horse programs don't self-replicate, but instead perform malicious actions, including compromising the confidentiality, integrity, or availability of the device or using its resources for malicious purposes.

Examples of mobile malware include the iPhoneOS.Ikee worm, which was targeted at iOS-based devices (for example, iPhones) or the Android.Pjapps threat, which enrolled infected Android devices in a hacker-controlled botnet.

## *Social Engineering Attacks*

Social engineering attacks, such as phishing, leverage social engineering to trick the user into disclosing sensitive information. Social engineering attacks can also be used to entice a user to install malware on a mobile device.

## *Resource Abuse*

The goal of many attacks is to misuse the network, computing, or identity resources of a device for unsanctioned purposes.  The two most common such abuses are the sending of spam emails from compromised devices and the use of compromised devices to launch denial of service attacks on either third-party websites or perhaps on the mobile carrier's voice or data network.

In the spam relay scenario, an attacker surreptitiously transmits spam emails to a herd of compromised devices and then instructs these devices to forward these emails over standard email or SMS messaging services to unsuspecting victims. The spam therefore appears to originate from legitimate mobile devices.

In the denial of service attack scenario, the attacker might instruct a large herd of previously-compromised devices to send a flood of network data (for example, network packets, SMS messages, etc.) to one or more targets on the Internet. Given the limited bandwidth available on today's wireless networks, such an attack could potentially impact the quality of either voice or data services on the wireless network in addition to impacting a targeted website.

## *Data Loss*

Data loss occurs when an employee or hacker exfiltrates sensitive information from a protected device or network. This loss can be either unintentional or malicious in nature. In one scenario, an enterprise employee might access their work calendar or contact list from a mobile device. If they then synchronize this device with their home PC, for example, to add music or other multimedia content to the device, the enterprise data may be unknowingly backed up onto the user's unmanaged home computer and become a target for hackers. In an alternative scenario, a user may access a sensitive enterprise email attachment on their mobile device, and then have their device stolen. In some instances, an attacker may be able to access this sensitive attachment simply by extracting the built-in SD flash memory card from the device.

## *Data Integrity Threats*

In a data integrity attack, the attacker attempts to corrupt or modify data without the permission of the data's owner.  Attackers may attempt to launch such attacks in order to disrupt the operations of an enterprise or potentially for financial gain (for example, to encrypt the user's data until the user pays a ransom fee). In addition to such intentional attacks, data may also be corrupted or modified by natural forces (for example, by random data corruption). For example, a malware program might delete or maliciously modify the contents of the mobile device's address book or calendar.

# Device Security Models

The designers of iOS and Android based their security implementations, to varying degrees, upon five distinct pillars:

- **Traditional Access Control:** Traditional access control seeks to protect devices using techniques such as passwords and idle-time screen locking.
- **Application Provenance:** Provenance is an approach where each application is stamped with the identity of its author and then made tamper resistant (using a digital signature). This enables a user to decide whether or not to use an application based on the identity of its author. In some implementations, a publisher may also analyze the application for security risks before publication, further increasing the pedigree of an app.
- **Encryption:** Encryption seeks to conceal data at rest on the device to address device loss or theft.
- **Isolation:** Isolation techniques attempt to limit an application's ability to access the sensitive data or systems on a device.

- **Permissions-based access control:** Permission-based access control grants a set of permissions to each application and then limits each application to accessing device data/systems that are within the scope of those permissions, blocking the applications if they attempt to perform actions that exceed these permissions.

Now that we've introduced the threat categories we wish to defend against and the five security pillars, the following sections provide a detailed security analysis of each mobile platform.

# Apple iOS

Apple's iOS operating system that powers iPod, iPhone, and iPad devices is effectively a slimmed down version of Apple's OS X Mac operating system. OS X is inherently a Unix-based system that traces its roots to NEXT corporation's Mach operating system, and ultimately to the FreeBSD variant of Unix.

While iOS leverages all five of the security pillars, its security model is primarily based on four of the five pillars: traditional access control, application provenance, encryption, and isolation. The next sections will cover these four primary pillars in detail, and then briefly discuss iOS's secondary reliance on permission-based access control.

## Traditional Access Control

iOS provides traditional access control security options, including password configuration options as well as account lockout options. For example, an administrator may choose the strength of the passcode and specify how frequently the user must update their passcode. They can also specify such items as the maximum number of failed login attempts before the device wipes itself.

### *How effective has Apple's Access Control implementation been?*

The access control features provided by iOS provide a reasonable level of security for the device's data in the event of loss or device theft. Essentially, iOS is at parity with traditional Windows-based desktops in this area.

## Application Provenance

Before fine art galleries sell expensive pieces of art, they make sure to verify the provenance and authenticity of these works. This gives the buyer confidence that they are obtaining an original work of quality and value. Apple employs a similar model with their iOS Developer and iOS Developer Enterprise programs. Before software developers can release software to iPhone, iPod, and iPad users, they must go through a registration process with Apple and pay an annual licensing fee. Developers must then "digitally sign" each app with an Apple-issued digital certificate before its release. This signing process embeds the developer's identity directly into to the app guarantees that the app author is an Apple-approved developer (since only these developers are issued such a certificate), and ensures that the app's logic cannot be tampered with after its creation by the author.

Today, Apple gives developers two different ways to distribute their applications to customers.

First, anyone wishing to sell their iOS app to the general public must do so by publishing their app on Apple's App Store. To post an app on the App Store, the software developer must first submit the app for certification by Apple—this certification process typically takes one to two weeks. Once an app has been certified, Apple posts it for sale on its App Store.*

Second, corporations wishing to deploy privately-developed apps to their internal workforce may register with Apple's iOS Developer Enterprise program. To be approved for this program, Apple requires that the applicant corporation be certified by Dun and Bradstreet, indicating that they're an established corporation with a clean track record. As a member of this program, enterprises may distribute apps developed in-house via an internal corporate website or by pushing the app using Apple's iOS management platform. As before, each app must be digitally signed by the enterprise before distribution to the internal workforce. Moreover, internally developed apps can only be used on devices on which the enterprise has installed a digital certificate called a "provisioning

* Apple has the ability to rapidly remove apps (that are found to be malicious or that violate their licensing agreement) from their App Store, but does not yet appear to posses an automated mechanism to remove malicious apps directly from iPhones/iPads once an app has been installed on the device.

profile". This certificate may be installed at the same time as the enterprise app, or in advance of the deployment of one or more enterprise apps. If the certificate is ever removed from the device or expires, then all apps signed with the certificate will cease to function.

While Apple explicitly permits corporations to distribute internal applications to their workforces, they prohibit sale/distribution of internally developed apps to third parties. If detected, this activity could lead to revocation of the enterprise's ability to participate in the iOS Developer Enterprise program. If such an abuse is detected, Apple presumably can simply issue a global revocation for the corporation's provisioning profile, immediately disabling all apps released by the vendor. This certificate requirement also enables a corporation to instantly disable its internally developed applications by simply removing the certificate from a device. This could be used, for example, to deprovision an employee's private device once the employee leaves the company.

The provenance approach employed by Apple certainly increases the odds that software developers will be held accountable for their applications and we believe that this has had a strong deterrent effect. However, it is by no means foolproof.  First, is certainly possible that a malware author could use a stolen identity to register for an account to sell malicious apps on the Apple App Store. Second, Apple does not discuss its app certification approach and it is possible that an attacker could slip malware past this certification process. On the positive side, Apple's requirement that all apps be digitally signed by Apple-approved software vendors does ensure that applications aren't tampered with, modified, or infected by hackers.

### *How effective has Apple's Application Provenance implementation been?*

The primary security goal of Apple's provenance approach is to limit malware, and in this regard, Apple has been effective. Thus far, we haven't seen actual malware targeting non-jailbroken iOS devices. Why is this? It is likely that malware authors steer away from the iOS platform because they understand that (A) they must register and pay to obtain a signing certificate from Apple, which makes it more likely they will get identified and prosecuted if they perform malicious activities, and (B) Apple tests each and every application that is submitted for publication on the App store for malicious behavior or violations of their policies, making it more likely that the attacker will be caught. Finally (C), Apple's code signing model prevents tampering with published apps—there is no way for an attacker to maliciously modify another app (for example, to add spyware to it) without breaking the "seal" on that app's digital signature.

It is important to note that Apple's provenance approach only applies to devices that have not been "jailbroken". Jailbroken devices—devices that have been intentionally hacked by their owners to give the owner administrative control over the device's operating system—have their provenance system disabled and may run apps from any source.  Such jailbroken devices have already been the target of at least two computer worm attacks (described in the iOS Malware section), and will likely be the target of increasing volumes of malware in the future.

## Encryption

The latest iPhones, iPads, and iPod Touch devices (that is, those using the iOS 4 operating system and beyond) employ a hybrid encryption model. First, iOS uses hardware-accelerated AES-256 encryption to encrypt all data stored in the flash memory of the device. Second, iOS protects specific additional data items, such as email, using an additional layer of encryption.

At first glance, iOS's full-device encryption approach would appear to offer a high degree of protection. However, Apple's implementation has a hitch. Since iOS runs background applications even when the user is not logged in to their device, and since these background applications need to access the device's storage, iOS needs to keep a copy of the decryption key around at all times so it can decrypt the device's data and provide it to these background apps. In other words, the majority of the data on each device is encrypted in such a manner that it can be decrypted without the need for the user to input the device's master passcode. This means that an attacker with physical access to an iOS device and with a functional jailbreak attack can potentially read most of the device's data without knowing the device's passcode.

In addition to hardware encryption, our research indicates that a small subset of iOS's data is secondarily encrypted in such a way that it may only be accessed if the device is unlocked via the user passcode. If the attacker doesn't have access to the device's passcode, then this data is essentially 100 percent secure while the device is

locked, whether or not an attacker has physical access to the device. Based on our research, iOS encrypts emails and attachments using this secondary level of encryption. Apple has indicated that other data may also be encrypted with this second level of encryption; however, we have not been able to verify this directly. Third-party applications can also manually leverage this encryption if they implement the required programming logic.

### *How effective has iOS's encryption implementation been?*

The main goal of encryption is to prevent loss of data due to device theft or loss. In this regard, Apple's encryption implementation may be considered a marginal success:

The main use case behind iOS's device-level encryption is rapid device wiping. Since every byte of data on the device is hardware encrypted with an encryption key, a device can be wiped by simply throwing away this key. If the encryption key is discarded, then all of the device's data is rendered inaccessible. This is exactly how Apple's device wiping technology works. Thus, if an administrator or user knows that a device has been lost or stolen early enough, they can almost certainly send a "kill signal" to the device via a third-party Mobile Device Management (MDM) solution and ensure that all of the data on the device is protected. Similarly, iOS devices can be configured to automatically throw away their hardware encryption key if the user enters an incorrect passcode too many times, rendering the data wholly unreadable.

However, a determined attacker that has physical access to a device and a functional jailbreaking tool can potentially obtain far more information. In a February 2011 report, German security researchers from the Fraunhofer Institute showed that using a six minute-long automated process, they could bypass the hardware encryption protections on an up-to-date, passcode-locked iPhone (running iOS 4.2.1) and obtain passwords and login information for most of the device's systems, including its Exchange email passwords and credentials, Wi-Fi passwords, VPN passwords, and voicemail passwords.[*] While most casual attackers won't have the sophistication required to launch this type of attack, this clearly shows that iOS's hardware encryption strategy is still vulnerable to attack.

Moreover, whether or not an iOS device is locked and in the user's pocket, or unlocked in their hand, apps running on the device may freely access iOS's calendar, contact list, photos (many of which are tagged with GPS coordinates), etc., since Apple's hardware decrypts this data on behalf of every running app. So should a malicious app bypass Apple's vetting process, or should an attacker compromise a legitimate app on the device (for example, by using a Web-based attack to compromise the Safari Web browser), the attacker could easily access and steal data from many of the device's systems.

## Isolation (Sandboxing)

The iOS operating system isolates each app from every other app on the system—apps aren't allowed to view or modify each other's data, logic, etc. One app can't even find out if another app is present on the device. Nor can apps access the iOS operating system kernel—they can't install privileged "drivers" on the device or otherwise obtain root-level (administrator) access to the device. This inherent design choice ensures a high degree of separation between apps, and between each app and the operating system.

All third-party applications running on iOS run with the same limited level of device control and are ultimately totally controllable by the iOS operating system and the user. For example, every third-party application running on an iOS device is subject to termination if the device is running low on available memory—no app can designate itself as "system critical" to avoid such termination by the operating system. The user may also terminate any app at any time with a few taps of the touchscreen. This is in contrast to PC-based applications that can easily install themselves into the operating system kernel and obtain an elevated privilege level to obtain total control of a system (and prevent easy termination by the user).

In addition to being isolated from each other and from the operating system kernel, applications are isolated from the phone's SMS and email in/out-boxes and email attachments within these mailboxes. Apps are also prohibited from sending SMS messages (without user participation) and from initiating or answering phone calls without the user's participation.

* http://www.sit.fraunhofer.de/en/Images/sc_iPhone%20Passwords_tcm502-80443.pdf

On the other hand, iOS apps are allowed to freely access the following system-level resources without any explicit granting of permission by the user. They may:

- Communicate to any computer over the wireless Internet.
- Access the device's address book including mailing addresses, notes associated with each contact, etc.
- Access the device's calendar entries.
- Access the device's unique identifier (a proprietary ID issued to each device by Apple).
- Access the device's phone number (this may be disabled via a simple configuration change by the user).
- Access the device's music/video files and its photo gallery.
- Access the recent safari search history.
- Access items in the device's auto-completion history.
- Access recently viewed items in the YouTube application.
- Access the Wi-Fi connection logs.
- Access the device's microphone and video camera.

## *How effective has iOS's application Isolation approach been?*

Application isolation is meant to address a number of different attacks, including preventing Web-based and network-based attacks, limiting the impact of malware, preventing malicious data loss, preventing attacks on the integrity of the device's data, and ensuring the availability of the device's services and data. Let's examine iOS's Isolation model on each:

### Web-based and network-based attacks

Since iOS isolates each app from every other app on the system, this means that if an attacker compromises an app, they will not be able to attack other apps or the iOS operating system itself (unless an unpatched vulnerability in iOS is attacked).

For example, consider Apple's Safari Web browser. If an attacker were to deliver an attack via a malicious Web page that took control of the browser's logic, this attack would be unable to spread to any other apps on the system beyond the browser, limiting its impact. However, this attack, once running in the Web browser process, could still access system-wide resources such as the calendar, the contact list, photos, the device's unique ID, etc., since these resources are available for access by all apps under the default iOS isolation policy. The malicious code could then exfiltrate this sensitive data to the attacker without his code having to ever escape the confines of the browser's sandbox. Moreover, resident malicious code within the browser process can also steal any data hosted in or that flows through the browser process itself, including Web passwords, credit card numbers, CCV security codes, account numbers, browsing history, bookmarks, etc. And such a malicious agent in the browser could also initiate malicious transactions on behalf of the user, without their consent.

So, in summary, iOS's isolation approach has thus far provided a great deal of protection against network-based attacks. However, attacks against specific apps like the Web browser, while being self-contained and blocked from impacting other apps, can still cause significant harm to a device.

### Limiting the impact of malware

While it is difficult to measure this empirically due to the small number of actual malware samples on iOS, iOS's isolation framework is theoretically effective at preventing classic malware attacks on the iPhone. Since apps can't access or modify other apps on the system, this prevents a malicious app from infecting or maliciously modifying other apps on the system, as a traditional parasitic computer virus might do. Further, the isolation layer prevents apps from installing operating system kernel drivers (such as kernel-based malware or root-kits) capable of running with the same administrator-level access as the operating system's kernel.

In this regard, iOS's isolation system has thus far been effective.

### Preventing Resource Abuse

iOS's isolation system can prevent a subset of resource abuse attacks. On the negative side, iOS apps are given unrestricted access to the Internet, so technically they could be used to launch email-based spam campaigns, search engine optimization campaigns (the attacker tricks a search engine into raising the ranking/visibility of

a particular website) and some types of denial of service attacks against websites or a carrier's network. However, it is important to note that we have never seen an actual example of such an attack.

On the positive side, given that iOS's isolation system prevents the automated transmission of SMS messages or automated initiation of phone calls, this eliminates the possibility of SMS-based DoS attacks, telephony-based DoS attacks, and SMS-based SPAM attacks on non-jailbroken devices.

### Preventing Malicious Data Loss

The isolation approach implemented by iOS completely prevents each app from accessing other apps' data—this policy is enforced regardless of whether apps encrypt their data or not, so long as the device has not been jailbroken. Moreover, beyond the library of media files, the calendar, and the contact database which are all accessible to any app, iOS has no centralized repository of shared data that might pose a serious compromise risk.

That said, these limited reservoirs of information (the calendar, media library, etc.) often store sensitive information, such as:

- Conference call numbers and passwords.
- Passwords for other systems (for example, bank accounts or enterprise logins).
- Credit card or bank account numbers that might be easily forgotten.
- Key codes for alarms and secure corporate offices.
- Employee names and phone numbers.
- Sensitive audio or video content, including internal audio and video pod-casts from senior management.

All of these items can obtained by any third-party app and exfilrated off the device over the Internet without any warning from the iOS's security systems.

### Preventing Attacks on the Integrity of the Device's Data

iOS's isolation policy allows apps to modify or delete the contents of the calendar and the contact list, but completely prevents modification or deletion of content from the user's media and photo libraries and from other device systems. While a malicious app could easily delete or modify all of the user's contacts and calendar entries, these can easily be recovered from a local backup (automatically created by iTunes during local syncs) or by synchronizing with a cloud-based data source like Exchange, MobileMe, or Google Calendar.

## Permissions-based Access Control

Apple has built a relatively limited permission system into iOS. Essentially, there are only four system resources that apps may access that first require explicit permission from the user. All other access to system services or data is either explicitly allowed or blocked by iOS's built-in isolation policy. Here are the permissions that an app may request:

- To access location data from the device's global positioning system.
- To receive remote notification alerts from the Internet (used by cloud-based services to send realtime notifications to apps running on a user's iPhone or iPad).
- To initiate an outgoing phone call.
- To send an outgoing SMS or email message.*

If an app attempts to use any of these features, the user will first be prompted for permission before the activity is allowed.

If the user grants permission to either the GPS system or the notification alert system, then the app is permanently granted access to these systems. In contrast, the user is prompted every time an app attempts to initiate an outgoing call or send an SMS message.

* Technically, iOS blocks local applications from using built-in iOS messaging systems to surreptitiously send SMS or email messages directly off the device without the user's consent. However, given that apps can connect to any other computer on the Internet without the user's express consent, apps can directly connect to Internet-based messaging services (for example, SMTP servers or SMS relay services) and then use these third-party services to send emails or SMS messages without the user's consent, effectively bypassing this permission-based protection system. It is important to note that SMS messages sent through one of these third-party services would not result in a charge to the user, which may have been Apple's primary goal: to prevent unauthorized sending of expensive text messages from a device.

### *How effective has iOS's permission system been?*

The GPS permission requirement prevents unwanted applications from surreptitiously tracking the user's location. To date, there have been no known attacks that have bypassed this protection on non-jailbroken devices.

The second push notification permission is not related to security but rather to ensuring battery life. The push notification subsystem in iOS frequently checks the Internet for new notifications, which can result in significant battery drain. Therefore, the effectiveness (or lack thereof) this aspect of the permission system has no impact on a device's security.

To our knowledge, these two permissions have effectively prevented attacks that attempt to surreptitiously initiate phone calls or send SMS messages (for example, to expensive pay services). As we'll see, Android's more lenient permission model has resulted in at least one such attack.

## Vulnerabilities

As of the time of this paper's writing, security researchers had discovered roughly 200 different vulnerabilities in various versions of the iOS operating system since its initial release. Of these, the vast majority of these vulnerabilities were of lower severity. Specifically, most would allow an attacker to take control of a single process (for example, the Safari process) but not permit the attacker to take administrator-level control of the device. The remaining handful vulnerabilities were of the highest severity, and when exploited, enabled an attacker to take administrator-level control of the device, granting them access to virtually all data and services on the device. These more severe vulnerabilities are classified as privilege escalation vulnerabilities because they enable an attacker to escalate their privileges and gain total control over the device.

While each of these vulnerabilities could have been targeted for malicious purposes, the majority of exploitation appears to have been initiated by device owners for the purpose of jail-breaking rather than as a means to maliciously compromise devices.

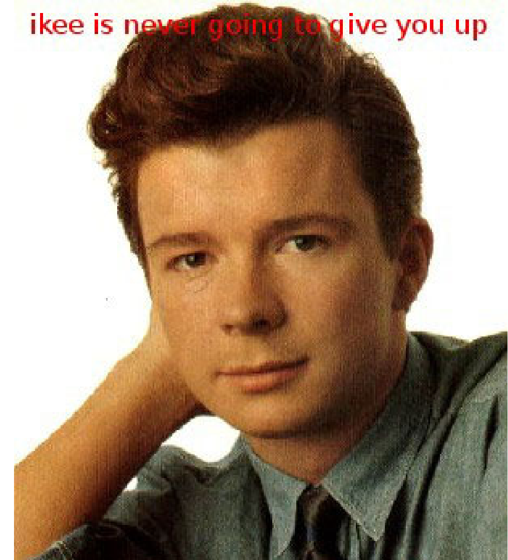
According to Symantec's data at the time this paper was authored, Apple took an average of 12 days to patch each vulnerability once it was discovered.

## Brief Overview of iOS Malware (and False Alarms)

**Aurora Feint (July, 2008):** This iPhone game uploaded contacts stored in iPhone's address book to the developer's servers in an unencrypted form. Apple briefly pulled this app from the Apple App Store, later restoring it after receiving an explanation from the developer. The developer explained that they used this contact data to match players up with their friends to enable over-the-air gameplay.

Figure 1

**iPhone.Ikee Wallpaper**




**Storm8 (November, 2009):** Storm8 Corporation released three games onto the Apple App Store that were downloaded by over twenty million users. These games transmitted the phone number from the iOS device to Storm8's servers for the purpose of uniquely identifying each user in their multiplayer game. Storm8 subsequently switched from using the device's phone number to using Apple's unique device ID value to identify users of its games.

**iPhoneOS.Ikee Worm (November, 2009):** This computer worm spread over-the-air (for example, across cellular and Wi-Fi networks) to jailbroken iOS devices, changing the device's background wallpaper to display a picture of 80's pop-star Rick Astley. iPhoneOS.Ikee performed no other malicious activity beyond changing the device's wallpaper. The worm was only capable of attacking devices that met three criteria: First, the device had to have been previously jailbroken by its owner. Second, the owner must have previously installed an SSH (secure shell) application on the device

(these applications typically enable a user to remotely connect to and control one computer or device from another computer). Third, the worm would only attack devices for which the default SSH password had not been changed.

**iPhoneOS.Ikee.B (November, 2009):** This computer worm also spread over-the-air to jailbroken iOS devices using the same SSH default password attack used by iPhoneOS.Ikee. Once the worm infected a new device, it would lock the screen and display the following text: “Your iPhone’s been hacked because it’s really insecure! Please visit doiop.com/iHacked and secure your iPhone right now!” In order to unlock an infected phone, the user was required to pay a €5 ransom to the attacker’s PayPal account.

Figure 2

**iPhone.Ikee.B Message**

## Summary of iOS Security

Overall, Symantec considers iOS’s security model to be well designed and thus far it has proven largely resistant to attack. To summarize:

- iOS’s encryption system provides strong protection of emails and email attachments, and enables device wipe, but thus far has provided less protection against a physical device compromise by a determined attacker.
- iOS’s provenance approach ensures that Apple vets every single publicly available app. While this vetting approach is not foolproof, and almost certainly can be circumvented by a determined attacker, it has thus far proved a deterrent against malware attacks, data loss attacks, data integrity attacks, and denial of service attacks.
- iOS’s isolation model totally prevents traditional types of computer viruses and worms, and limits the data that spyware can access. It also limits most network-based attacks, such as buffer overflows, from taking control of the device. However, it does not necessarily prevent all classes of data loss attacks, resource abuse attacks, or data integrity attacks.
- iOS’s permission model ensures that apps can’t obtain the device’s location, send SMS messages, or initiate phone calls without the owner’s permission.
- None of iOS’s protection technologies address social engineering attacks such as phishing or spam.

# *Android*

Android is a marriage of the Linux operating system and a Java-based platform called Dalvik, which is an offshoot of the popular Java platform. Essentially, software developers write their apps in the Java programming language and then using Google tools convert their resulting Java programs to run on the proprietary Dalvik platform on Android devices. Once converted, such an app can run on any Android device. It is unclear why Google chose to use a non-standard Java platform to run its apps; perhaps this approach was taken to avoid patent infringement.

Each Android app runs within its own virtual machine (just as Java applications do), and each virtual machine is isolated in its own Linux process. This model ensures that no process can access the resources of any another process (unless the device is jailbroken). While Java’s virtual machine was designed to be a secure, “sandboxed” system capable of containing potentially malicious programs, Android does not rely upon its virtual machine technology to enforce security. Instead, all protection is enforced directly by the Linux-based Android operating system.

Android’s security model is primarily based on three of the five security pillars: traditional access control, isolation, and a permission-based security model. However, it is important to note that Android’s security does not simply arise from its software implementation. Google releases the programming source code for the entire Android project, enabling scrutiny from the broader security community. Google argues that this openness helps to uncover flaws and leads to improvements over time that materially impact the platform’s level of security.*

* This claim appears to be true—there have been less than two-dozen vulnerabilities discovered in the Android platform since it’s release, an extremely low number.

The next three sections will explore Android's use of these primary pillars, while the following two sections will then examine Android's secondary reliance upon the Provenance and Encryption approaches.

## Traditional Access Control

Android 2.X versions provide rudimentary password configuration options, including the ability to specify the strength of the device passcode, specify the phone's lockout time span, and specify how many failed login attempts must occur before the device wipes its data. Android 3.0 also introduces the notion of password expiration, enabling administrators to compel users to update their password on a regular schedule.

### *How effective has Android's Access Control implementation been?*

Android password policy system is sufficient to protect devices against casual attacks. However, since current versions of Android do not encrypt data stored on the removable SD memory card (for example, the 16- or 32-gigabyte memory chip used to store data and multimedia files), an attacker with physical access to an Android device could simply eject the SD memory card and obtain a subset of the device's data in a matter of seconds, bypassing any and all password controls enabled on the device.

## Isolation

Like iOS, Android employs a strong isolation system to ensure that apps only access approved system resources. This isolation system not only isolates each app from other apps on the system, but also prevents apps from accessing or modifying the operating system kernel, ensuring that a malicious app can't gain administrator-level control over a device. The default isolation policy prohibits access to virtually every subsystem of the device, with the following noteworthy exceptions:

- Apps may obtain the list of apps installed on the device and examine each application's programming logic (but not its private data).
- Apps may read (but not write to) the contents of the user's SD flash card, which typically holds the user's music, video files, installed programs, and possibly documents or saved attachments. Apps may read all of the data on the SD card without restriction (regardless of which app created a particular piece of data, all apps can read that data).
- Apps may launch other applications on the system, such as the Web browser, the maps application, etc.

Of course, this default isolation policy is so strict that it inhibits the creation of many classes of applications. As such, Android permits applications to request á-la-carte access to the device's other subsystems—the isolation system then enforces each app's expanded set of permissions. This á-la-carte access model is discussed in the next major section below.

### *How effective is Android's isolation system?*

When we consider Android's isolation system, we must evaluate its ability to (A) limit the damage in the situation where an attacker manages to compromise a legitimate app (for example, a Web browser), and (B) its ability to block or constrain traditional malware.

First, since Android isolates each app from every other app on the system, from most of the device's services, and from the operating system itself, this means that if an attacker compromises a legitimate app, they will not be able to attack other apps or the Android operating system itself. This is a positive of Android's isolation model.

Let's consider Android's Web browser. Web browsers are by far the most targeted class of legitimate application, since attackers know that Web browsers often have security flaws that can easily be exploited by a properly crafted malicious Web page. Imagine that an attacker posted a malicious Web page that attacked a known flaw of Android's Web browser. If an unsuspecting user surfed to this Web page, the attack could inject itself into the Android browser and begin running.

Once running in the Web browser's process, would this attack pose a threat? Yes and no. First, Android's isolation policy would ensure that the attack could not spread beyond the browser to other apps on the system or to the operating system kernel itself. However, such an attack could access any parts of the system that the Web

browser app had been granted permission to access. For example, if the Web browser had permission to save or modify data on the user's SD storage card (for example, to save downloads on the card), then the attacker could take advantage of this permission to corrupt data on the SD storage card. Therefore, an attacker effectively gains the same control over the device as the app they manage to attack, with varying implications depending on the set of permissions requested by the compromised app.

Moreover, malicious code within the attacked process can also steal any data that flows through the process itself. In the case of a Web browser, the attack could easily obtain login names, passwords, credit card numbers, CCV security codes, account numbers, browsing history, bookmarks, etc. Since mobile users often access internal enterprise applications via their mobile Web browser, this could lead to leakage of highly sensitive enterprise data, even if VPN or SSL encryption are employed. And such a malicious agent in the browser could also initiate malicious transactions on behalf of the user, without their consent.

Next let's consider the ability of Android's isolation system to protect against malicious apps such as Trojan horses and spyware. Since Android's isolation system is designed to isolate each app from other apps on the system, this ensures that a malicious app can't tamper with other apps on the system, access their private data or access the Android operating system kernel. However, given that each app can request permission to access other device subsystems such as the email inbox, the GPS system, or the network, it is possible for such a malicious app to operate within the confines of Android's isolation system and still conduct many categories of attacks, including resource attacks, data loss attacks, etc. Ultimately, as we'll see in the next section, the reliance upon the (potentially uninformed) user to grant a set of permissions to an app is the weak link in Android's isolation approach.

Attackers have in a small number of instances bypassed Android's isolation system by exploiting flaws in its implementation (that is, vulnerabilities). However, the number of vulnerabilities in Android has generally been small in number, and most have been fixed quickly (our Vulnerabilities section covers this in more detail).

## Permissions-based Access Control

By default, most Android applications can do very little without explicitly requesting permission from the user to do so. For example, if an app wants to communicate over the Internet, it must explicitly request permission from the user to do this; otherwise the default isolation policy blocks it from initiating direct network communications.

Each Android app therefore contains an embedded list of permissions that it needs in order to function properly. This list of requests is presented to the user in non-technical language at the time an app is installed on the device, and the user can then decide whether or not to allow the app to be installed based on their tolerance for risk. If the user chooses to proceed with the installation, the app is granted permission to access all of the requested subsystems. On the other hand, if the user chooses to abort the installation, then the app is completely blocked from running. Android offers no middle ground (allowing some permissions, but rejecting others).

Third-party apps can request permission to use the following high-level subsystems:

- **Networking subsystems:** Apps can establish network connections with other networked devices over Wi-Fi or using the cellular signal.
- **Device identifiers:** Apps can obtain the device's phone number, the device ID (IMEI) number, its SIM card's serial number, and the device's subscriber ID (IMSI) number. These codes can be used by criminals to commit cellular phone fraud.
- **Messaging systems:** Apps can access emails and attachments in the device's inbox, outbox, and SMS systems. Apps can also initiate transmission of outgoing emails and SMS messages without user prompting and intercept incoming emails and SMS messages.
- **Calendar and Address book:** Apps can read, modify, delete, and add new entries to the system calendar and address book.
- **Multimedia and image files:** Apps may access multimedia (for example, MP3 files) and pictures hosted by the device's photo application.

- **External memory card access:** Apps can request to save, modify, or delete existing data on external plug-and-play SD memory cards. Once granted this permission, apps have unrestricted access to all of the data on the SD card, which is not encrypted by default.
- **Global positioning system:** Apps may obtain the device's location.
- **Telephony system:** Apps can initiate and potentially terminate phone calls without the user's consent.
- **Logs and browsing history:** Apps may access the device's logs (such as the log of outgoing and incoming calls, the system's error log, etc.) as well as the Web browser's list of bookmarks and surfing history.
- **Task list:** An app may obtain the list of currently running apps.

### *How effective is Android's permission system?*

At first glance, Android's permission system seems to be extremely robust, enabling software vendors to limit an application to the minimal set of device resources required for operation. The problem with this approach is that ultimately, it relies upon the user to make all policy decisions and decide whether an app's requested combination of permissions is safe or not. Unfortunately, in the vast majority of cases, users are not technically equipped to make these security decisions. In contrast, Apple's iOS platform simply denies access, under all circumstances, to many of the device's more sensitive subsystems. This increases the security of iOS-based devices since it removes the user from the security decision-making process. However, this also constrains each application's functionality, potentially limiting the utility of certain classes of iOS apps.

For example, consider a video game which requests permission to access the Internet and also to access the device's identification numbers. It's difficult for a novice user to determine whether this combination of privileges is dangerous or not. In a legitimate scenario, the app might use the device's unique ID to look up the user's high scores on a server. Yet by requesting these same two privileges an app could also export the device's IMEI and IMSI numbers to an attacker—both of these device identification numbers could be used by criminals to commit wireless fraud. For example, an IMEI number stolen from a working phone can be used to unlock a stolen phone that was previously disabled by the carrier. The typical user has no basis to understand the implications of granting a particular set of permissions, and many benign-looking combinations of permissions can be used to launch an attack.

So far, we've seen only a handful of different malware apps released for Android, but it's already clear that many are able to cause damage without having to "crack" or bypass Android's permission system. Each malicious app simply requests the set of permissions it needs to operate, and in most cases, users happily grant these permissions on the promise of playing the next great video game or using an up-and-coming calendar organizer. Android's isolation system then happily grants the app full access to the requested set of device services.

By requesting the proper permissions, a malicious app could launch resource abuse attacks (for example, sending large volumes of spam or launching distributed denial of service attacks), data loss attacks (stealing data from the device's calendar, contact list, etc.), and perform data availability/integrity attacks (by modifying/deleting data in calendar, contact list, or on the SD card). So, to conclude, while Android implements a robust permission system, its dependence on the user and its leniency in offering access to most of the device's sub-systems compromise its effectiveness and have already opened up Android devices to attack.

## Application Provenance

Whereas Apple's iOS platform is built upon a strong application provenance model, the provenance approach adopted by Google for Android devices is less rigorous and consequently, less secure.

### *Android's Digital Signing Model*

The ultimate goal of digitally signing an application is twofold: one, to ensure that the app's logic is not tampered with, and two, to allow a user of the app to determine the identity of the app's author. Google's approach undermines both of these goals.

Why is this? Like Apple, the Android operating system will only install and run apps that have been properly signed with a digital certificate. However, unlike Apple, software developers need not apply to Google to obtain a code-signing certificate. Instead, application developers can generate their own signing certificates, as often as they like, without any oversight. In fact, the software developer can place any company name and contact

information in their certificate that they like, for example “Author=Dr. Seuss”. The result is that a malware author can generate “anonymous” digital certificates as often as they like and none of these certificates or malware signed with them can be traced back to the author.

In order for developers to sell their apps on Google’s official Android Marketplace, developers must pay a $25 fee via credit card. This enables Google to associate the payee with the digital certificate used to digitally sign the developer’s apps and should act as a mild deterrent against malware authors posting malware on the Android Marketplace (if they use their own credit card to register). However, given that developers have the ability to distribute their apps from virtually any website on the Internet—not just the Android marketplace—malware programs can also be distributed with anonymity without any vetting by Google.

This approach has two problems. First, it makes it much easier for malware authors to create and distribute malicious applications since these applications can’t be tracked back to their source. Second, this approach makes it easier for attackers to add Trojan horses to existing legitimate apps. The attacker can obtain a legitimate app, add some malicious logic to the app, and then re-sign the updated version with an anonymous certificate and post it onto the Internet. While the newly signed app will lose its original digital signature, Android will certify and install the newly signed malicious app with its anonymous digital signature. Thus, Android’s model does not realistically prevent tampering.

### *Android’s Vetting Model*

While Apple enforces a single vetting and distribution channel for all iOS apps, Google chose a far more open model for Android apps. First, Google does not appear to perform a rigorous security analysis of applications posted onto its Android Marketplace. This means that malware authors can distribute their apps through this distribution channel with less likelihood of being discovered. While Apple’s certification approach can certainly fail to detect some classes of attacks, it at least acts as a deterrent to malware authors. In contrast, Google’s lack of validation offers less of a deterrent.

Second, Android application developers can distribute their apps from virtually any website on the Internet—they are not limited to distributing their apps via the Android Marketplace. While, by default, Android devices may only download applications from Google, users may override this setting with a few taps of their touch-screen and then download apps from virtually anywhere.* This allows users to “side-load” apps from any source on the Internet.

Like iOS, Android will never silently install an app onto a device. The user is always notified before a new application is installed (with a few notable exceptions, described below). This prevents drive-by attacks common on PCs and requires attackers to employ social engineering to trick users into agreeing to install malicious apps on their devices.

### *How effective is Android’s provenance approach?*

History shows us that platforms that allow software developers to anonymously release their applications have experienced larger volumes of malware than those platforms that require each app to be digitally stamped with the certified identity of its author. The Android platform appears to reinforce this historical precedent.

During 2010 and 2011, attacks such as Android.Rootcager, Android.Pjapps, and Android.Bgserv all took advantage of weaknesses in Android’s provenance model. In each of these cases, the attacker appropriated an existing, legitimate application, stripped the original, legitimate digital signature from the application, injected malicious code into the application, re-signed the Trojanized application using an uncertified digital signature, and then distributed the app via either the official Android Marketplace or third-party websites. In all, these threats impacted hundreds of thousands of users.

Since attackers can effectively generate their own digital certificates as frequently as they like and use them to sign malware, we argue that this compromises the value of Android’s provenance system, especially for apps distributed outside Android’s App Marketplace where there’s no software developer vetting process. Moreover,

* Some wireless carriers forbid such “side-loading” of apps from third-party websites.

we argue that since no single authority evaluates/verifies all Android apps, attackers are more likely to release attacks without worry of getting caught—this too, we believe, has led to an increase in the prevalence of Android malware.

## Encryption

As of the time of this writing, only the latest generation of Android tablet devices (running Android 3.0) support hardware encryption to protect data. Unfortunately, at this time Google has not disclosed how this encryption works, making it difficult to determine its strengths and weaknesses. However devices running earlier versions of Android (including virtually all Android-based mobile phones available at the time this paper was authored) rely upon the isolation model, instead of encryption, to protect data such as passwords, user names, and application-specific data. This means that if an attacker is able to jailbreak a device or otherwise obtain administrator-level access to a device by exploiting a vulnerability or by obtaining physical access to a device, they can access virtually every byte of data on the device, including most of the passwords, Exchange/private email account credentials, etc.[*]

As with iOS, third-party Android applications may optionally encrypt their data using standards-based encryption algorithms, but application developers must explicitly add this logic to their program. Otherwise, all data created by applications is saved in an unencrypted form.

## Vulnerabilities

As of the time of this paper's writing, security researchers had discovered 18 different vulnerabilities in various versions of the Android operating system since its initial release. Of these, most were of lower severity and would only allow an attacker to take control of a single process (for example, the Web browser process) but not permit the attacker to take administrator-level control of the device. The remaining few vulnerabilities were of the highest severity, and when exploited, enabled an attacker to take root-level control of the device, granting them access to virtually all data on the device.

To date, all but four of these eighteen vulnerabilities have been patched by Google. Of the four unaddressed vulnerabilities, one is of the more severe privilege escalation type. This vulnerability has been addressed in the 2.3 release of Android, but has not been fixed for prior versions of the operating system. Given that most carriers have not updated their customer's phones from Android 2.2 to 2.3, this means that virtually every existing Android phone (at the time of this writing) is currently open to attack. This vulnerability may be exploited by any third-party app and does not require the attacker to have physical access to the device. As an example, the recent Android.Rootcager and Android.Bgserv threats both leveraged this vulnerability to obtain administrator-level control of devices. Even more interestingly (and controversially), Google's fix tool for Android.Rootcager also had to exploit this vulnerability in order to circumvent Android's isolation system to remove parts of the threat from the device.

According to Symantec's data at the time this paper was authored, Google took an average of eight days to patch each vulnerability once it was discovered.

## Brief Overview of Android Malware

**Android.Pjapps / Android.Geinimi (January/February, 2010):** These threats were designed to steal information from Android devices and enroll the compromised device in a botnet. Once enrolled, these Trojans enabled the attacker to launch attacks on third-party websites, steal additional device data, deliver advertising to the user, cause the user's phone to send expensive SMS messages, etc. To distribute these threats, the attackers obtained existing legitimate programs from the Android store, injected the malware logic into them and then distributed these modified versions on third-party Android marketplace websites. Users download what they thought were popular, legitimate applications without knowledge of the extra malicious payload included in the packages.

* In a few cases, Android does store "authentication tokens" rather than passwords to prevent loss of passwords. These authentication tokens are numeric values derived from the original password using a one-way hashing function, making it impossible to obtain the user's original password, while still enabling a login from the device.

**AndroidOS.FakePlayer (August, 2010):** This malicious app masquerades as a media player application. Once installed, it silently sends SMS messages (at a cost of several dollars per message) to premium SMS numbers in Russia. Devices connected to wireless carriers outside of Russia are unaffected since the SMS messages are not properly delivered; however, this threat illustrates how easy it is to steal funds from unsuspecting users.

**Android.Rootcager (February, 2011):** Also known as Android.DroidDream, this attack was similar in nature to the Android.Pjapps attack—the attacker infected and redistributed more than 58 legitimate applications on Google's App Market. Hundreds of thousands of users were infected, tricked into thinking they were downloading legitimate applications. Once installed by the user, the threat attempted to exploit two different vulnerabilities in Android to obtain administrator-level control of the device. The threat then installed additional software on the device, without the user's consent. The software exfiltrates a number of confidential items, including: device ID/serial numbers, device model information, carrier information, and has the ability to download and install future malware packages without the user's knowledge (this is only possible since the threat exploited a vulnerability to bypass Android's isolation model). Both vulnerabilities used by Android.Rootcager were patched in Android's 2.3 release; however, most Android-based devices on the market are running earlier Android versions as of this paper's writing, meaning that most devices are still susceptible to this style of attack.

**Android.Bgserv (March, 2011):** In response to the Android.Rootcager threat, Google deployed a tool over-the-air to clean up infected Android devices. Shortly after this cleanup tool was released, attackers capitalized on the hype and released a malicious fake version of the cleanup tool. This Trojan horse exfiltrates user data such as the devices IMEI number and its phone number to a server in China.

## Summary of Android's Security

Overall, while we believe the Android security model is a major improvement over the models used by traditional desktop and server-based operating systems, it has two major drawbacks. First, its provenance system enables attackers to anonymously create and distribute malware. Second, its permission system, while extremely powerful, ultimately relies upon the user to make important security decisions. Unfortunately, most users are not technically capable of making such decisions and this has already led to social engineering attacks. To summarize:

- Android's provenance approach ensures that only digitally signed applications may be installed on Android devices. However, attackers can use anonymous digital certificates to sign their threats and distribute them across the Internet without any certification by Google. Attackers can also easily "trojanize" or inject malicious code into legitimate applications and then easily redistribute them across the Internet, signing them with a new, anonymous certificate. On the plus side, Google does re-

Figure 3

AndroidOS.FakePlayer Permissions

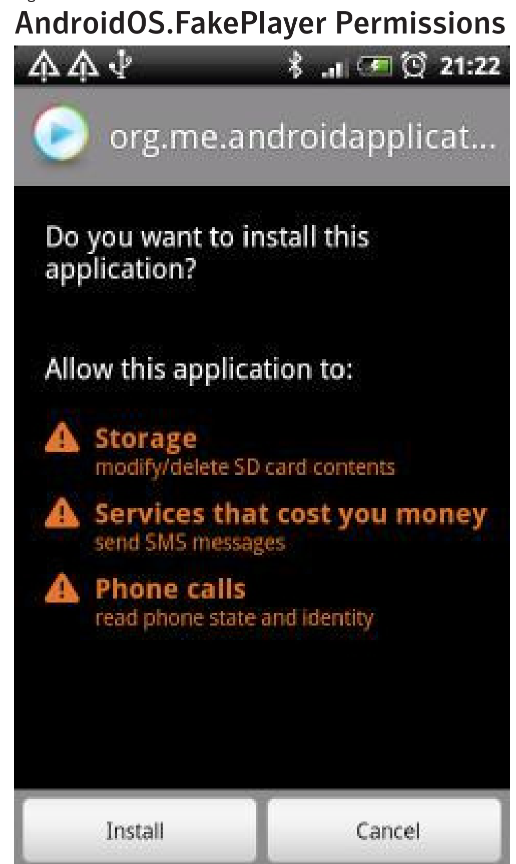
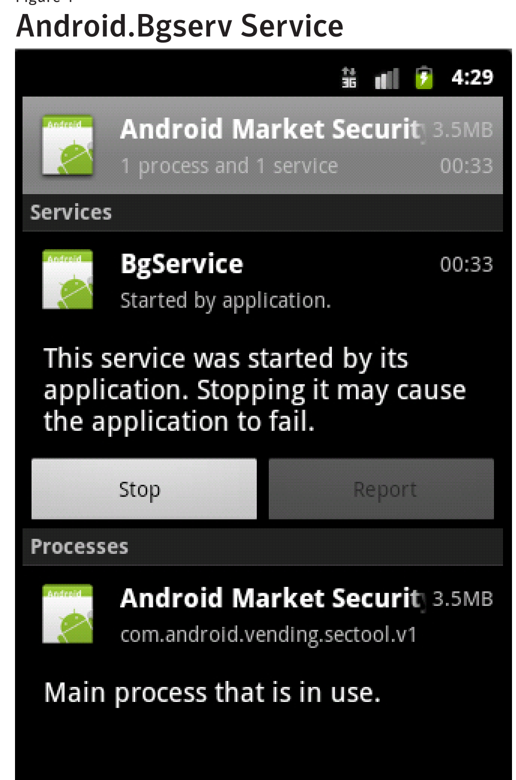



Figure 4

Android.Bgserv Service



quire application authors wishing to distribute their apps via the official Android App Marketplace to pay a fee and register with Google (sharing the developer's digital signature with Google). As with Apple's registration approach, this should act as a deterrent to less organized attackers.

- Android's default isolation policy effectively isolates apps from each other and from most of the device's systems including the Android operating system kernel, with several notable exceptions (apps can read all data on the SD card unfettered).
- Android's permission model ensures that apps are isolated from virtually every major device system unless they explicitly request access to those systems. Unfortunately, Android ultimately relies upon the user to decide whether or not to grant permissions to an app, leaving Android open to social engineering attacks. Most users are unequipped to make such security decisions, leaving them open to malware and all of the secondary attacks (for example DDoS attacks, Data Loss attacks) that malware can launch.
- Android recently began offering built-in encryption in Android 3.0. However, earlier versions of Android (running on virtually all mobile phones in the field), contain no encryption capability, instead relying upon isolation and permissions to safeguard data. Thus, a simple jailbreak of an Android phone or theft of the device's SD card can lead to a significant amount of data loss.
- As with iOS, Android has no mechanism to prevent social engineering attacks such as phishing attacks or other (off-device) Web-based trickery.

Table 1

**Resisting attack types**

| Resistance to: | Apple iOS | Google Android |
|---|---|---|
| Web-based attacks | Full Protection | Full Protection |
| Malware attacks | Full Protection | Little Protection |
| Social Engineering attacks | Little or No Protection | Little or No Protection |
| Resource Abuse/ Service attacks | Good Protection | Moderate Protection |
| Data Loss (Malicious and Unintentional) | Moderate Protection | Little Protection |
| Data Integrity attacks | Moderate Protection | Little Protection |

# iOS vs. Android: Security Overview

Tables one and two summarize our conclusions about the various strengths and weaknesses of both the iOS and Android mobile platforms.

Table 2

**Security feature implementation**

| Security Pillar | Apple iOS | Google Android |
|---|---|---|
| Access Control | Good Protection | Moderate Protection |
| Application Provenance | Full Protection | Little Protection |
| Encryption | Good Protection | Little Protection |
| Isolation | Moderate Protection | Full Protection |
| Permission-based Access Control | Moderate Protection | Moderate Protection |

**Legend**

- Full Protection
- Good Protection
- Moderate Protection
- Little Protection
- Little or No Protection

# Device Ecosystems

Today's iOS and Android devices do not work in a vacuum—they're almost always connected to one or more cloud-based services (such as an enterprise Exchange server, Gmail, MobileMe, etc.), a home or work PC, or all of the above. Users connect their devices to the cloud and to PC/Mac computers in order to:

- Synchronize their enterprise email, calendars, and contacts with their device.
- Synchronize their private email, calendars, and contacts, and other digital content with their device (for example, music and movie files).
- Back up their device's email, calendars, contacts, and other settings in case their device is lost.

When properly deployed, both Android and iOS platforms allow users to simultaneously synchronize their devices with multiple (private and enterprise) cloud services without risking data exposure between these clouds. However, these services may be easily abused by employees, resulting in exposure of enterprise data on both unsanctioned employee devices as well as in the private cloud. As such, it is important to understand the entire ecosystem that these devices participate in, in order to formulate an effective device security strategy.

In a typical deployment, an employee connects their device to both an enterprise cloud service such as an Exchange server that holds the employee's work calendar, contacts, and email, as well as to a private cloud service, such as Gmail or Apple's MobileMe, which holds their private contacts, calendar events, and email.

Once a device is connected to one or more data sources, both iOS and Android provide a consolidated view of both corporate and private email, calendars, and contact lists that unifies data from both services into one seamless user interface, while internally maintaining a layer of isolation for the data from each service. In such a sanctioned deployment, this isolation ensures that no enterprise data finds its way onto the private cloud servers and vice versa (in other words, a work meeting entered into the employee's work calendar will never be synchronized into the user's private Gmail calendar, and a private appointment entered in the user's Gmail calendar would never find its way into the Exchange server).

This isolation prevents loss of enterprise data and also enables administrators to safely wipe enterprise data from a device while letting the user retain their own private data should they leave the company.

While such an enterprise-sanctioned deployment isolates data from each source, ensuring that enterprise data is not inadvertently synchronized with the employee's private cloud, it is quite easy for users to use third-party tools or services to intentionally or unintentionally expose enterprise data to third-party cloud services, unmanaged computers and devices. Here are the most common scenarios:

## Scenario #1: Unsanctioned Desktop Sync

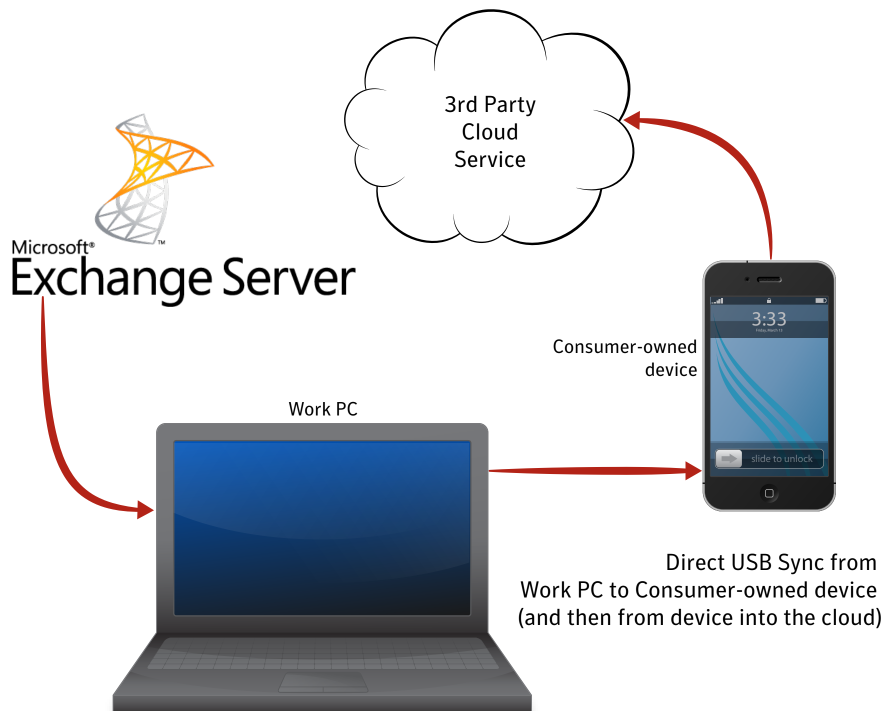
In this scenario, the employee brings in a consumer device that is not under enterprise management nor approved to hold enterprise data. In order to synchronize the user's work calendars/contact lists with the device, the employee uses either iTunes or a third-party synchronization tool to directly synchronize contacts and calendar entries from their PC/Mac to their device. This approach effectively migrates all of the user's enterprise calendar/contact data from the Exchange server via the local Outlook or iCal client on the user's work computer into the employee's private device.

Once the employee has used this approach to synchronize their work calendars/contacts to their enterprise device, they may opt to enroll their device in a third-party cloud service, such as Gmail or Apple's MobileMe service to make their calendar/contacts available via any Web browser (for example, so the employee, or perhaps a spouse can view their schedule online). Services like Gmail and MobileMe are able to directly pull data off the user's device into the cloud.

This type of employee behavior results in exposure of enterprise data to private clouds which do not have enterprise-level governance or protections (for example, password strength requirements), potentially opening up this data to attackers. In addition, this type of activity frequently exposes enterprise data to the employee's home computer (as we'll see below).

Figure 5

**Scenario #1: Unsanctioned Desktop Sync**

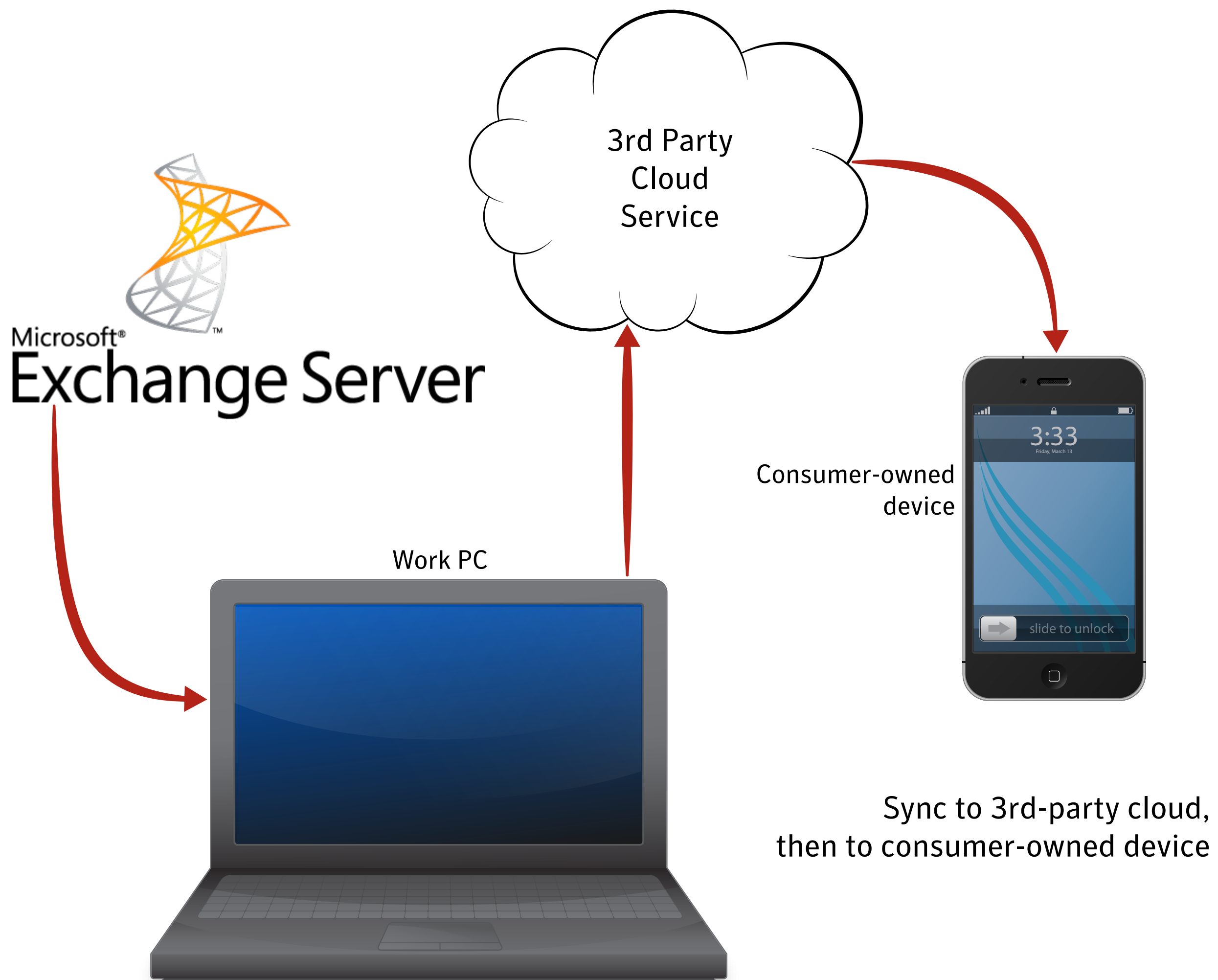


## Scenario #2: Unsanctioned Enterprise Desktop to Cloud Sync

In this scenario, the employee installs a tool such as Google Calendar Synch or iTunes on their work PC/Mac and uses this to synchronize their data directly with an unsanctioned private cloud, such as Gmail or MobileMe. Such synchronization programs automatically synchronize the desktop calendar and contacts from Outlook or iCal with a cloud service. This effectively migrates all of the user's enterprise calendar/contact data from the Exchange server, via the user's desktop, into an uncontrolled cloud. Next, the employee configures their device to synchronize directly with the private cloud provider, causing the device to download the employee's enterprise calendars and contacts. The employee is therefore able to obtain their work data on their device without having to connect the device directly to the enterprise Exchange server or to the employee's work PC.

Again, this type of activity exposes private enterprise data to both a third-party private cloud as well as to the employee's consumer device—both of which are not governed or secured by the enterprise.

Figure 6
**Scenario #2: Unsanctioned Enterprise Desktop to Cloud Sync**



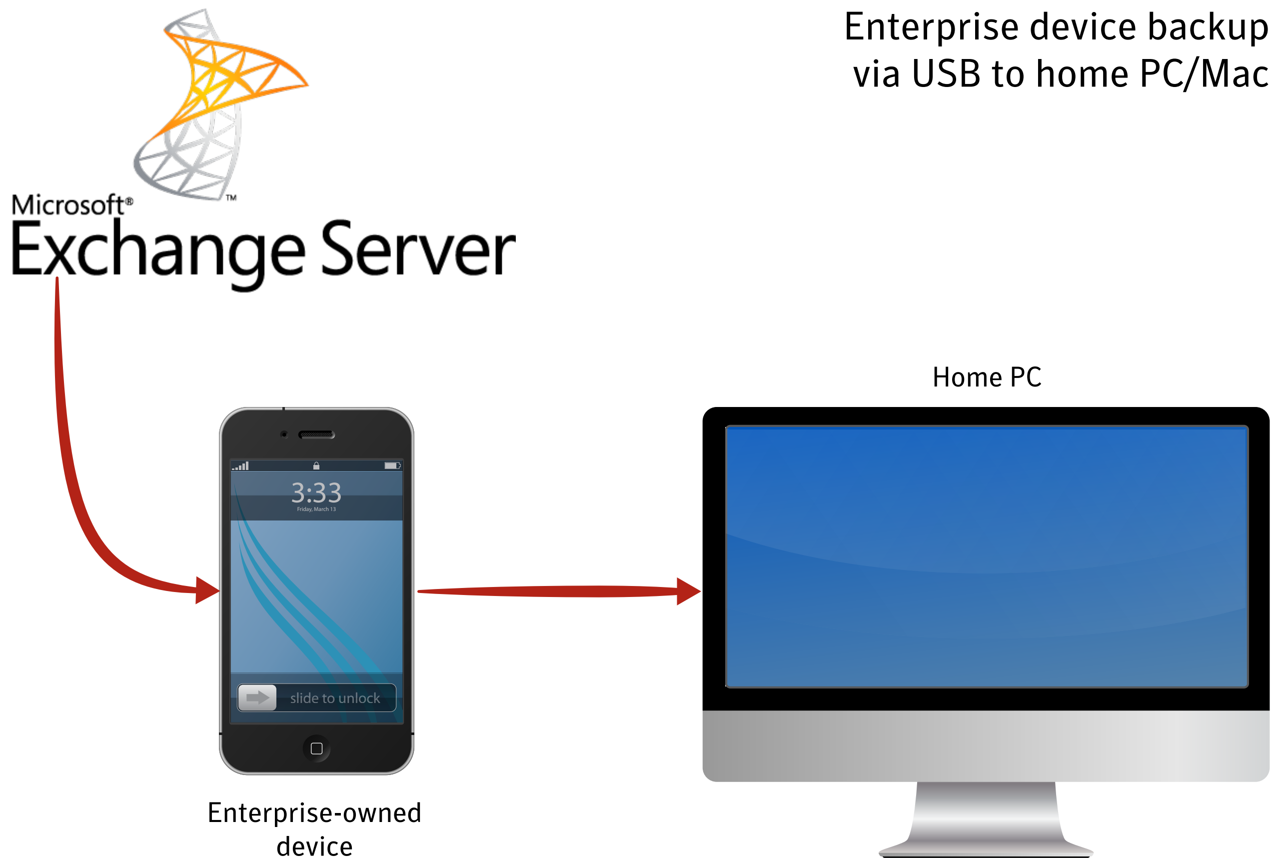

## Scenario #3: Unsanctioned Enterprise Device Sync with Home PC

Users regularly synchronize their home and work devices with their home PC/Mac to transfer music and multi-media files and to synchronize their device's calendars and contact lists with their private calendar/contact lists on their home computer. Employees with an iPhone use the standard iTunes software to perform this synchronization, whereas Android users need to leverage a third-party package.

Even if the user only chooses to synchronize music and multimedia files between their device and their home computer (opting not to synchronize calendars or contacts), the synchronization software will make a backup of the device's data to the computer in case a problem occurs. This backup typically contains the entire calendar and address book, notes, as well as device settings. Importantly, this data may not necessarily be encrypted when backed up to the user's home computer. For example, iTunes, by default, does not password-protect or encrypt the backup stored on the user's computer. This has the unintended consequence of migrating the employee's enterprise-owned data onto the their home PC in an unencrypted form even in cases where the employee opted not to explicitly synchronize enterprise calendar or contact data with their home computer.

Figure 7

**Scenario #3: Unsanctioned Enterprise Device Sync with Home PC**




# Mobile Security Solutions

We expect the nascent market for mobile security solutions to develop quickly. The nature of these security solutions will be largely driven by the evolving threat landscape and the constraints imposed by the security models of each platform.

Some of the initial mobile security approaches we have observed so far include:

## *Mobile Antivirus*

There are already a number of first-generation antivirus scanners for the Android platform. However, given iOS's strict isolation model, it is impossible to implement an iOS-based antivirus scanner for iOS-based devices (without relaxation of the isolation system by Apple). These scanners are effective at detecting known Android threats, but provide little protection against unknown threats. Ultimately Symantec expects traditional scanners to be replaced by cloud-enabled, reputation-based protection.

Of the six different categories of threats that face mobile devices, mobile antivirus solutions can address threats in the malware category as well as a subset of malware-based attacks in the resource abuse, data loss, and data integrity categories.

## Secure Browser

Several companies have introduced their own secure Web browser apps for both iOS and Android platforms. These apps are meant to be used instead of the built-in Web browsers provided on the iOS and Android platforms. Each time the user visits a URL in the secure browser, the browser checks the URL against a blacklist or reputation database and then blocks any malicious pages. The only problem with this secure browser approach is that the user cannot use the familiar, factory-installed Web browser shipped with the device. Instead, they must use the third-party secure Web browser to do all surfing.

Of the six different categories of threats that face mobile devices, secure browsers can effectively address Web-based attacks and social engineering attacks. The secure browser can also potentially block the introduction of malware downloaded through the browser.

## Mobile Device Management (MDM)

These tools enable the administrator to remotely administer managed iOS and Android devices. Typical security policies might include setting the password strength, configuring the device's VPN settings, specifying the screen lock duration (how long before the screen locks and a password is required to unlock the device), or disabling specific device functions (like access to an App marketplace) to prohibit potentially risky behaviors. In addition, the administrator may perform security operations like wiping lost or stolen devices, or using the device's onboard geo-location service to locate a device.

While mobile device management solutions don't specifically protect against any explicit threat category, they can help to reduce the risk of attack from many of the categories. For example, if the administrator uses the MDM solution to configure the device to block introduction of all-new apps, this can eliminate the introduction of new malware, and also limit resource abuse, integrity threats, and some intentional or unintentional data loss.

## Enterprise Sandbox

Sandbox solutions aim to provide a secure sandbox environment where employees can access enterprise resources such as email, calendar, contacts, corporate websites, and sensitive documents. All data stored in the sandbox, and data transmitted to and from services accessible via the sandbox, is encrypted. To use the sandbox, the user must first log in and the sandbox must then check with a corporate server to ensure that the user is still authorized to access both local data (on the device) as well as enterprise services. Given that the sandbox is provisioned by the corporate administrator, it can easily be deprovisioned if the device is lost or stolen. This approach has the effect of dividing the device's contents into two zones: a secure zone for the enterprise data, and an insecure zone for the employee's personal and private data. The benefit of such a solution is it enables the consumer to use their own device, yet it still lets them safely access enterprise data. The drawback of such a device is that the user can't use the regular mail, calendar, or contact apps built into the device to access enterprise resources, forcing them to adapt to a different set of sandboxed, potentially less usable equivalent apps. This may compel some employees to bypass the sandbox and use unsanctioned means to access enterprise resources.

The enterprise sandbox is primarily focused at protecting enterprise assets, such as enterprise documents, access to the enterprise intranet, enterprise emails and calendar events, etc., from attack. Therefore the sandbox approach is focused on preventing malicious and unintentional data loss. While this approach doesn't actually block the other six attack categories explicitly, it does implicitly limit the impact of these attacks on enterprise assets.

### *Data Loss Prevention (DLP)*

These tools scan the publicly accessible storage areas of each device for sensitive materials. Due to iOS's isolation system, iOS-based DLP tools can only inspect the calendar and contact lists for sensitive information. On Android, such a tool could scan the external flash storage (that is, the SD card), the email and SMS inboxes, as well as the calendar and contact lists. These products would be unable to scan the data of any other apps on the system, such as document viewers, word processors, spreadsheets, third-party email clients, etc., due to the isolation models of both iOS and Android. Thus, DLP solutions are not able to detect all sensitive data stored on or flowing through these devices.

## Conclusion

Today's mobile devices are a mixed bag when it comes to security. On the one hand, these platforms have been designed from the ground up to be more secure—they raise the bar by leveraging techniques such as application isolation, provenance, encryption, and permission-based access control. On the other hand, these devices were designed for consumers, and as such, they have traded off their security to ensure usability to varying degrees. These tradeoffs have contributed to the massive popularity of these platforms, but they also increase the risk of using these devices in the enterprise.

Increasing this risk is the fact that employees bring their own consumer devices into the enterprise and leverage them without oversight, accessing corporate resources such as calendars, contact lists, corporate documents, and even email. In addition, employees often synchronize this enterprise data with third-party cloud services, as well as their home PC. This back door connectivity results in the loss of potentially sensitive enterprise data across third-party systems that are out of the enterprise's direct control and governance.

To conclude, while mobile devices promise to greatly improve productivity, they also introduce a number of new risks that must be managed by enterprises. We hope that by explaining the security models that undergird each platform, and the ecosystems these devices participate in, we've provided you, the reader, with the knowledge to more effectively derive value from these devices and also more effectively manage this risks they introduce.

Symantec™
Security Response

Any technical information that is made available by Symantec Corporation is the copyrighted work of Symantec Corporation and is owned by Symantec Corporation.

NO WARRANTY . The technical information is being delivered to you as is and Symantec Corporation makes no warranty as to its accuracy or use. Any use of the technical documentation or the information contained herein is at the risk of the user. Documentation may include technical or other inaccuracies or typographical errors. Symantec reserves the right to make changes without prior notice.

**About Symantec**

Symantec is a global leader in providing security, storage and systems management solutions to help businesses and consumers secure and manage their information. Headquartered in Moutain View, Calif., Symantec has operations in more than 40 countries. More information is available at www.symantec.com.

**About the author**

Carey Nachenberg is a Vice President in Symantec's Security, Technology, and Response organization and a Symantec Fellow.

For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 (800) 745 6054.

Symantec Corporation
World Headquarters
350 Ellis Street
Mountain View, CA 94043 USA
+1 (650) 527-8000
www.symantec.com

Copyright © 2011 Symantec Corporation. All rights reserved. Symantec and the Symantec logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.