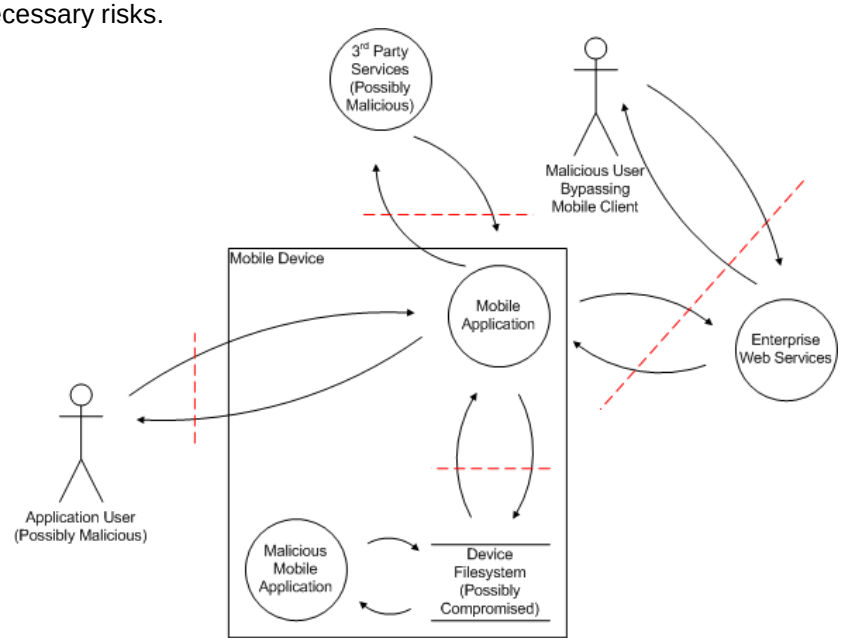
Secure Mobile Application Development Reference

DENIM GROUP

Secure Mobile Application Development Reference

Overview

Mobile and smartphone applications have very different threat models than their web-based counterparts. As more organizations explore ways to push functionality to mobile devices there will be a desire to move an increasing amount of sensitive onto the device. In addition, there will be a push to have more sensitive calculations performed on devices. Developers need to both understand the capabilities of their chosen development platform(s) as well as understand how to design and build applications to securely take advantage of mobile capabilities without exposing their organizations or application users to unnecessary risks.



Developers building mobile applications need to understand the threat model for the system they are building as well as understanding that the mobile application itself is only part of the system that attackers will attempt to compromise. Input that crosses a trust boundary should be positively validated and should not be used to make critical security decisions. Also, developers must be careful about what data is stored on the device because devices may be stolen or otherwise fall into unauthorized hands. Access permissions for local files and databases are also important because device owners might unwittingly install other application on the device that are malicious. Network communications can be sniffed and potentially modified in transit, so care must be taken when communicated sensitive data to and from the device.

Secure architecture and design principles can be useful when beginning the development of a new application so that possible concerns are known up-front. The recommendations drawn from these design exercises must then be implemented during development, and often the implementation of these requirements has platform-specific concerns. This document attempts to provide summary information of the overall design concerns as well as platform-specific recommendations. In addition, links to other resources are provided so that developers can find additional reference information when required.

In addition to this Reference, there are a number of resources available for developers interesting in creating secure mobile applications.

- Denim Group Secure Mobile / Smartphone Development Site
- OWASP Mobile Security Project
- Veracode Mobile App Top 10 List

Overview of Application Development

Every mobile application platform has different characteristics and so developers must be familiar with the specifics for their platform if they are going to begin developing applications. Also, development of mobile applications typically requires applications to be initially developed and run on an emulator before being tested on actual mobile devices.

- What languages can applications be written in?
- How are they deployed?

iOS (iPhone/iPad)	Android
iPhone and iPad applications are typically written in Objective-C and compiled to ARM machine code. All developers can run applications in a local emulator and developers with who have paid $99 USD/year to enroll in the iOS Developer Program can deploy applications to their phones via the USB cable. For actual production application installation (on non-jailbroken phones) the applications must be downloaded from Apple's iTunes Store. • Apple site for their iOS Developer Program	Android applications are written in Java and the Java source code is compiled to Dalvik Executable (DEX) binaries that are run on the Dalvik virtual machine. Developers can run applications in a local emulator and can also install applications on the device and debug them via a USB connection. To debug via USB, USB debugging must be enabled on the device, and the application must be declared as being debuggable in the AndroidManifest.xml file. Production applications can either be loaded onto Android phones via a USB connection or device SD card and can be downloaded from Google's Application Store. Loading applications via USB is known as "side loading" and also requires that the system setting of "Unknown Sources" to be unchecked in the Settings > Applications configuration. • Android documentation on Using Hardware Devices

Overview of Secure Development

Just as every mobile platform has different characteristics, they also have different security features, capabilities and weaknesses that developers should know about before beginning development.

- What are good sources of information for developers concerned about overall platform security as well as jumping-off places to use to search for more detailed information?

iOS (iPhone/iPad)	Android
Apple provides a Secure Coding Guide with iOS-specific recommendations on both the security features of the platform (HTTPS, keyring, etc) as well as how to develop secure features (input validation, avoiding buffer overflows, etc) • Apple's Security Concepts Overview • Apple's Secure Coding Guide for iOS	Google runs a Google Group for Android security discussions and there are some other resources available on the Internet. • Google Group for Android Security Discussions • iSec Partners released a guide for Developing Secure Mobile Applications for Android • Penn State Systems and Internet Infrastucture Security Laboratory presentation and example applications Understanding Android's Security Framework

Defeating Platform Environment Restrictions

Mobile device platforms are typically shipped so that root-level access to the platform is not immediately available to device owners. Researchers have devised ways to bypass these restrictions for most platforms, allowing power users and malicious attackers to gain greater access to the device, either to install arbitrary applications or to inspect or modify system-level attributes.

- How can device platform restrictions be bypassed?
- What are common ways to bypass platform restrictions?

iOS (iPhone/iPad)	Android
iOS devices can be "jailbroken." This allows access to the device as the root user and also allows 3rd party applications to be installed. • Wikipedia article on Jailbreaking iOS devices • Wikipedia article on the iPhone Dev Team, a group of researchers who have done extensive work on iOS jailbreaking	Android devices can be "rooted." This allows access to the device as the root user and allows for modification of the core Android system. In addition, rooting a device can allow a developer to install custom kernels on most devices. Unlike iOS devices, Android device do not need to be "rooted" in order to allow the installation of 3rd party applications that do not come from the Android store; it is possible to install arbitrary 3rd party application APKs on normal Android devices. • Wikipedia article on Rooting Android devices • Homepage for CyanogenMod, a common Android aftermarker firmware

Installing Applications

Mobile applications must be installed onto devices before being available to testers and users. Also, most mobile devices either allow or require production users to install applications onto their devices from one or more application stores.

- How are applications installed?
- What sort of verification and protection does the App Store provide?

iOS (iPhone/iPad)	Android
Non-jailbroken iOS devices can only install applications from the official Apple iTunes App Store. The App Store has an application approval process whose methods are not publicly disclosed, but that does not appear to do any meaningful security checking of applications. Instead applications are checked for the use of undocumented APIs or other violations. Apple can disable installed applications via updating a blacklist that the device periodically checks. • Apple's App Store Submission Tips • Gizmodo article on Apple's ability to remotely disable applications	Applications for Android phones are typically installed via the Google Marketplace. Application APK files can also be copied onto the device and installed manually. • Bright Hub's How To Install and Remove Applications on Google Android Phones • Bright Hub's How to Install APK Files on Your Google Android Phone

Application Permissions Model

Mobile devices contain sensitive information such as email messages, user contacts and the device owner's current location. In addition, mobile devices have access to sensitive capabilities such as the ability to make phone calls and send SMS messages. Because applications on these mobile devices are often developed by untrusted 3rd party developers, mobile application platforms typically have features that limit access to this senstive information and senstive capabilities unless the user allows an application access. Developers should be careful to only ask for permissions that are required for them to accomplish their specific application goals in order to potentially limit damage if their applications are compromised.

- What is the basic permissions model for applications running on the platform?
- What do they have access to and how do they gain access to sensitive device capabilities such as text messaging and GPS locations?

iOS (iPhone/iPad)	Android
When applications attempt to use APIs that require access to sensitive resources (such as the current location or the camera), iOS confirms that access should be allowed via a pop-up. Interestingly there is no confirmation required for apps to begin recording audio. • StackOverflow discussion of accessing GPS on iPhone • Overview information on Permissions from programming4.us	Each application is installed under its own Linux user account, thus isolating it from other applications on the system via Linux file and account permissions. Furthermore, application access to sensitive device resources such as the GPS location or phone calling is guarded by permissions enforced by the Davlik virtual machine. These permissions are laid out in the AndroidManifest.xml file and are confirmed by the user at application install time. • Android documentation on Using Permissions • Android documentation for the <permission> tag in AndroidManifest.xml files

Local Storage

Mobile devices have the ability to store information in files, databases and other constructs. Because devices can be lost or transferred to other users without being wiped, application developers should be very careful about storing sensitive informaiton locally on the device. Avoiding storing senstive information on the device is preferable because then the risk of compromise is minimized.

- Where can applications store local data on the device?
- What formats are allowed?

iOS (iPhone/iPad)	Android
Applications are given access to their own portion of the iOS filesystem that is within the application sandbox and inaccessible to other applications. Files can be designated for Sharing and such files are accessible in the Documents/ directory in iTunes. Files can also be marked as Protected so that they can only be accessed when the device is unlocked. Property List (plist) files can be used to store user preferences and other configuration information in a way that can be moved between OS X and iOS applications. • Apple overview page on iOS Data Management • Apple information about File and the Filesystem on iOS • Apple information about Shared files • Apple information about Protected files • Apple's Introduction to Property Lists	Android applications have a variety of local storage options. They can store files in both internal storage that will be protetected by the default Android/Linux permissions model that segregates access to application files via Linux file/group permissions or external storage on an SD card that will not be covered by those protections. Unless there are special circumstances, files shoudl be created with Context.MODE_PRIVATE or Context.MODE_APPEND, which will use Linux permissions to make them readable and writable only to the application that created the file (and the root user on rooted devices). Files that are created using the Context.MODE_WORLD_READABLE can be read by other applications and should not be used to store data that a malicious application should not have access to. Files that are created using the Context.MODE_WORLD_WRITABLE can be written to by other applications and data read from these files should not be trusted. In addition, Android applications can create SQLite databases for storing application information. Also, Shared Preferences can be used to store key/value data. Finally, Content Providers can be used to store data for a given application as well as for sharing with other applications. • Android documentation on Data Storage • Android Javadoc for Context.openFileOutput() describing file permission options

Encryption APIs

As mentioned above, it is preferable not to store any sensitive information on a device because of the risk of compromise. If sensitive data must be stored on the device, it should be encrypted to prevent disclosure. However, storing encrypted data on the device is challenging because of key storage issues; a device that contains both encrypted informations as well as the key required to recover that encrypted information can easily be compromosed by a reasonably-determined attacker. In addition, it should be expected that captured devices will be rooted or jailbroken so that attackers can access information and run code that might not be allowed by the plastform running under normal conditions.

- What encryption libraries are available from the native device API?
- What 3rd party encryption libraries are available?
- Are their known limitations to the available encryption libraries?
- How can sensitive information stored on the device best be protected?
- How do these protections hold up for captured devices or devices that have been rooted or jailbroken?

iOS (iPhone/iPad)

iOS provides access to a variety of certificate and key management functions so that applications can access various encryption capabilities. In addition, iOS provides applications access to a Keychain service that allows the application to securely store local data such as passwords and encryption keys. Applications can access their Keychain items but other applications are not allowed access. Items stored in the keychain can also be stored such that they can only be recovered when the device has been unlocked with the PIN. Items stored without PIN protection can be recovered from jailbroken iPhones, so sensitive data should only be stored in the keychain combined with PIN protection. However, even with keychain protection, phones with numeric or easy-to-guess PINs may be susceptible to brute force attacks (for example there are only 10,000 possible 4-digit numeric PIN options), so extremely sensitive data should probably never be stored on the phone. Also, the Open Source SQLCipher extension to the SQLite database engine can be used to encrypt SQLite database files with AES 256.

- Apple documentation of Certificate, Key and Trust Services
- Apple's example CryptoExercise code
- Keychain Services Concepts in the iOS Reference Library
- SANS Blog Post How Not to Store Passwords in iOS
- SQLCipher iPhone Application Tutorial
- Research demonstrating recovery of non-PIN-protected Keystore items

Android

Android provides access to industry-standard encxryption APIs via the javax.crypto libraries. Also, some organizations have chosen to use the Bouncy Castle Java libraries with success.

- Android javax.crypto Javadocs
- Main Bouncy Castle site

Network Communications

Most "interesting" mobile applications will not run completely on the mobile device. In addition to operations on the device, applications often need to access network-attached resources. Mobile platforms offer a variety of networking options including provider networks, WiFi and others. Developers should take note not to send sensitive data over unencrypted connections because it might be intercepted by attachers. In addition, developers may want their applications to determine what networks they area attached to before sending certain information.

- What libraries are available for application to communicate over the network?
- What protocols are natively supported?

iOS (iPhone/iPad)	Android
iOS provides access to BSD sockets which provides the basis for accessing higher-level protocols. In addition, iOS provides dedicated libraries to access more advanced networking options such as Bonjour, gaming peer-to-peer and HTTP/HTTPS. • Apple overview page on Networking and Internet • O'Reilly iPhone SDK chapter on Network Programming	Android provides access to the standard java.net.* classes as well as a number of Apache HTTP Utilities (via the org.apache.http package) and Android-specific networking classes to access sockets, HTTP and HTTPS connections as well as SIP and Wifi (on supporting devices). • IBM Developerworks article on Networking with Android • Android Javadoc for standard java.net package • Android Javadoc for Apache org.apache.http package • Android Javadoc for Android android.net package • Android Javadoc for Android android.net.http package • Android Javadoc for Android android.net.sip package • Android Javadoc for Android android.net.wifi package

Protecting Network Communications

As mentioned above, sensitive data shoudl not be sent across network connections unencrypted. Fortunately, most mobile platforms offer access to at least basic encrypted communication mechanisms such as SSL sockets and HTTPS web requests. Care should be taken to force SSL connections to use appropriately strong encryption and to properly verify the identify of the connected server.

- What encryption methods are supported for network communications?

iOS (iPhone/iPad)	Android
iOS provides implementations of commonly-required transport-layer security protocols. • Apple developer guidance on Secure Transport	Android provides access to the javax.net.ssl classes that allow developers to use SSL/TLS to protect network communications. Developers using the android.net.SSLCertificateFactory helper class should understand that the available options to turn off server validation will open up the possibility of man-in-the-middle attacks. • Android javax.net.crypto Javadocs • Android android.net.SSLCertificateSocketFactory Javadocs

Native Code Execution

Running native code opens up opportunities for the introduction of whole classes of vulnerabilities such as buffer overflows and format string attacks. Whenever possible, developers should utilize managed code because it typically provides automatic memroy management and array bounds checking. Also, some platforms offer access to buffer overflow protection technologies such as non-executable stacks and address space layout randomization. If it is required to run native code these protections should be used.

- How can attackers attempt to exploit buffer overflows and other native-code-execution vulnerabilities?
- What protections are available for the platform?

iOS (iPhone/iPad)	Android
iOS applications are written in Objective-C and compiled to ARM machine code, so all application code is subject to potential buffer overflow vulnerabilities. iOS makes the stack non-executable by default, but has yet to implement ASLR (although the antid0te post-jailbreak software can provide ASLR for iOS). • Overview information on Exploit Mitigation for iOS from programming4.us • Main site for antid0te post-jailbreak ASLR implementation	Android applications are typically written in Java and compiled into bytecode that runs on the Dalvik virtual machine. Android also supports the exectuion of native code via the Native Development Kit (NDK) which allows applications running in the Dalvik virtual machine to make JNI-style calls to native code. This is typically done either to implement routines that require native code speed for performance reasons, or in order to reuse an existing body of C/C++ code without rewriting it in Java. From a security standpoint, native code is risky because it runs without the typical protections of the Dalvik virtual machine such as automated memory management and array bounds overflow detection. • Android Overview of the Native Development Kit (NDK) • Android documentation for the Native Development Kit (NDK)

Application Licensing and Payments

Most mobile platforms provide some capability for developers to be paid for licensing their application as well as for applications to allow users to make payments through the application. Understanding the capabilities and limitations of these parts of the mobile application platform will allow best protect their applications from piracy.

- How are applications licensed for the platform?
- Are their known weaknesses in the licensing system?

iOS (iPhone/iPad)	Android
iOS application purchase and licensing is typically handled via the Apple iTunes App Store. Apple also offers the capability to make In App Purchases. • Introductory video from Apple in In App Purchases • Apple App Store Quick Reference on Getting Started in In App Purchase	Android offers a runtime library that queries the Google Market to determine licensing policies for an application. This licensing service offers a flexible model for determining if the use of an application meets the licensing criteria as well as how the application should respond to licensing violations. Android is developing an in-app Billing service that allows apps to initiate purchases of digital content. • Google Android guidance on Licensing Your Applications • Early-access documentation for Android In-App Billing • Security and Design best practices for Android in-app Billing

Mobile Browser

Mobile platforms rely on platform-provided browsers to access the web. These are either used as standalone applications, or are embedded into custom applications in order to provide access to content and functionality. Many attacks on mobile platforms have used the browser as a vector, so developers need to understand how their mobile platform makes use of the included browser.

- What browser or browsers run on the mobile platform?
- What rendering engine is in use?

iOS (iPhone/iPad)	Android
iOS uses a mobile version of the Safari browser, which is based on the WebKit HTML rendering engine. • Main WebKit website • Apple documentation on WebKit plugins	Android's default browser uses the WebKit HTML rendering engine and a version of Chrome's V8 JavaScript engine. • Main WebKit website • Wikipedia documentation on Android features

Browser URL Handling

Most mobile platforms allow developers to register their applications to handle requests and content initially handled by the device web browser. This allows developers to provide a richer experience than a web browser on its own could provide, but also opens up avenues for attackers to try and subvert application behavior by seeding malicious websites with specially-crafted links intended to execute a target application, but with malicious parameters. Developers should understand the situations under which their applications might be executed and be sure to properly validate incoming data and request appropriate confirmation from application users before performing sensitive actions.

- Does the mobile platform browser allow applications to link URL protocols to applications - either default or 3rd party?
- Are their known weaknesses to avoid when creating applications that will act as URL protocol handlers?

iOS (iPhone/iPad)	Android
iOS allows applications to register to handle different URL schemes. When Safari attempts to process a URL with a registered scheme, it will launch the registered application and call its "handleOpenURL:" method. Registration is accomplished by an application listing the desired URL scheme in its Info.plist file. • Apple URL Scheme Reference for built-in URL schemes in the Safari Reference Library • SANS Blog Post Insecure Handling of URL Schemes in Apple's iOS • Code samples for using built-in and 3rd party iPhone URL Schemes • Index of known URL Schemes for iOS	Android allows applications to register to handle events raised by the browser for specified protocols. This is done by setting an <intent-filter> in the AndroidManifest.xml file with a <data> tag and android:scheme attribute for the desired scheme/protocol to be handled. In addition, the <intent-filter> must also include <category android:name="android.intent.category.BROWSABLE" /> to inform the system that the Activity is safe to call via the Browser application. • Android documentation on Intents and Intent Filters • URLHandler code example on GitHub

Mobile Application SMS/Push Update Handling

Most mobile platforms provide a way for applications to register for messages that might arrive when they are not in the foreground. Applications should treat these messages as untrusted input and validate them before use.

- Does the mobile platform allow applications to register for "push" type events?
- What are the different methods for this registration and what are the security characteristics of each?

iOS (iPhone/iPad)	Android
iOS allows applications to register for local and push notifications so that applications not in the foreground can present messages to users and allow users to optionally launch the application. Because these messages can come from potentially malicious sources, all input from them should be positively validated before being used by the application. • Local and Push Notification Programming Guide • Specific information about the Security Architecture of push notifications	The "official" way for Android applications to recieve "push"-type messages is the Cloud to Device Messaging Framework (C2DM). As with all external inputs presented to an application, these messages should be positively validated before being used by the application. Other approaches to providing this type of functionality have emerged. In all cases, however, messages should be treated as untrusted input and should be validated before use. • Android documentation for the Cloud to Device Messaging Framework

Contact Denim Group for more information about building secure mobile applications and testing the security of mobile applications. Contact Dan Cornell (dan _at_ denimgroup _dot_ com) with questions, comments or suggestions about this document.