

Sponsor: ESE Capstone
Dept. No.: G021 / E54A
Contract No.: W15P7T-12-C-F600
Project No.: 031280SE-D2

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation.

This document was prepared for authorized distribution only. Approved for Public Release: 12-3459. Distribution Unlimited

©2011 The MITRE Corporation.
All rights reserved.

|  |
| --- |
| MTR120076 |
| MITRE TECHNICAL REPORT |

Android Secure Application Development Guidance for DoD

Michael Peck, Shawn Valle

30 September 2011

Prepared By:

***Signed***

Robert “Pat” Benito, JC2 Pilot Task Lead 2/19/12

Approved By:

***Signed***

Josiah R. Collens, Jr. 2/20/12
Director of Integration for Joint C2
NSEC

This page intentionally left blank.

Executive Summary

Android applications developed for US Department of Defense (DoD), are required to go through a workflow process to evaluate and test for meeting expected Cyber Security and Information Assurance guidelines. Applications that meet the evaluation guidelines can be permitted into the enterprise application market, known as CAPStore, for user distribution. The following documentation identifies the technical requirements and guidance Android application developers should adhere to when developing applications for DoD.

The details within are technical and security focused, and should be made available to software engineers and IA engineers. The material is organized with a logical flow in mind, initially focusing on application permissions, then into securing code and data, and finally focusing on multiple application interaction.

Table of Contents

1 Introduction 1-1

2 Application Permissions 2-1

2.1 Leverage Android Permissions Model 2-1

2.2 Creating New Manifest Permissions 2-1

3 General Application Authentication 3-1

3.1 Password Guidance 3-3

4 Data Protection 4-1

4.1 Database Encryption 4-1

4.2 SD Card Storage 4-1

4.3 Android Application Package 4-2

4.4 File Permissions 4-2

5 Follow Secure Programming Practices 5-1

5.1 Input Validation 5-1

5.2 Avoiding SQL injection attacks 5-1

5.3 Avoiding command injection attacks 5-2

5.4 Sign Application Packages 5-2

5.5 Avoid Android NDK or Java JNI Use, Unless Necessary 5-3

5.6 Third-Party Libraries 5-3

6 Secure Data Communication 6-1

6.1 Leverage TLS/SSL 6-1

6.2 Parameter Content 6-4

7 Secure Inter-App Communication 7-1

7.1 Securing Android Intents 7-1

7.2 Securing Content Providers 7-4

8 Application Update Process 8-1

9 Non-Android SDK Applications 9-1

9.1 Browser-based Apps 9-1

9.2 Adobe Air Apps 9-1

List of Figures

Figure 3‑1 Potential Sample Authentication 3-2

Figure 3‑2 Standard DoD PED Banner 3-4

This page intentionally left blank.

# Introduction

This document is targeted towards developers of Android mobile applications and those with Information Assurance responsibilities over mobile deployments. This document’s purpose is to ensure that Cyber Security and Information Assurance best practices are incorporated into the application development process.

Android applications and mobile devices have many challenges within DoD Information Assurance (IA) IT systems. Generally, a DoD IT system must apply for IA certification & accreditation (C&A) through one of the following documented processes: DoD Information Assurance Certification and Accreditation Process (DIACAP) or Platform IT (PIT). Both have stringent guidance programs which must be addressed in order to receive an authority to operate (ATO) with a DoD Designated Approving Authority (DAA).

Systems that do not have connectivity to the Global Information Grid (GIG) can apply as PIT with their DAA. If accepted, the PIT system has a lower level of guidance to follow for IA. Most systems, however, have some connection to the GIG, and fall under the IA category of DIACAP.

Systems that plan to follow DIACAP should assign an IA Manager (IAM) to address, guide, and ensure all programmatic DIACAP instructions and directives are adequately met. Along with the IAM, an IA Technical (IAT) engineer should be engaged for the lifecycle of the project to work as an integral part of an Integrated Project Team (IPT), as well as the IA liaison to the IAM.

Programs adhering to DIACAP should review documents DoD Directive 8500.1 and DoD Instruction 8500.2 for identification of artifacts necessary for C&A. From an engineering perspective, the first level of technical security guidance can be found in Defense Information Systems Agency (DISA) Security Technical Implementation Guides (STIGs). In the context of Android applications, operating systems, devices, and communications, the challenges are great. DISA has released a draft STIG for Android, but the STIG's focus is on email and calendar collaboration tools using the Good Mobile Messaging suite and Dell Android 2.2 devices. The finalized Android STIG is scheduled for October 2011. Projects using other Android devices or using Android devices for broader purposes need to request additional STIGs from DISA. DAA’s leverage DISA STIG’s when assessing a systems level of technical IA. DIACAP related document DoDI 8500.2 identifies STIG’s as an example of security guidance for engineering. The ultimate goal is for DoD information systems to maintain the appropriate level of confidentiality, integrity, authentication, non-repudiation, and availability.

As a general rule, the IPT determines which STIGs directly apply to its project. In the case of Android applications, we look to guidance from “Application Security and Development STIG Version 3 Release 2”, released 10 October 2010 and “Draft Android 2.2 (Dell) STIG Version 1 Release 0.1”, released 17 June 2011[[1]](#footnote-1). Along with relevant STIGs, industry best practices for application development on the Android platform are leveraged; including Google’s Android documentation, and multiple security engineering print resources.

# Application Permissions

Android implements an application permission framework that provides the ability to control the allowed operations of individual applications. This section provides guidance on making appropriate use of Android’s permission framework.

## Leverage Android Permissions Model

This section identifies guidance for application permissions. It addresses how an application obtains access to capabilities of the Android device, and how a developer should grant and document the necessity for each granted access.

Android application permissions are expressed in the application’s AndroidManifest.xml file. Typically, the user is shown the list of requested permissions at install time, though this varies based on the particular mechanism used to install the application.

Do not anticipate Android users will understand the technical vernacular. Avoid using *Binder, Activity,* or *Intent* when describing permissions to users.[[2]](#footnote-2)

When applying any android.permission, the developer shall provide external documentation as to the need/nature of the permission. As a rule, the requested permissions shall be limited to those required for the application to perform its intended tasks. An example of a permissions entry in the AndroidManifest.xml follows. In this example, the application requests the ability to receive SMS messages.

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

 package="com.android.app.myapp" >

 <uses-permission android:name="android.permission.RECEIVE\_SMS" />

 ...

</manifest>

As of this writing, there are 116 Manifest permissions documented in the Android Developers SDK.[[3]](#footnote-3) See Appendix A for all permissions and details.

## Creating New Manifest Permissions

If an application requires access to another application, a new permission may be defined. Our guidance recommends that developers define new permissions for controlling inter-application access. More details are provided in Section 7.

Any newly defined manifest permission shall be documented with rationale of the need. If the purpose of a custom permission is to reduce several individual permission messages, each of the intentionally bypassed permissions shall be documented. The documented manifest permission rationale will be included as an artifact for certification & accreditation.

# General Application Authentication

This section leverages DISA’s draft Android STIG and the Application Security and Development STIG to provide guidance for application authentication and access control.

DISA guidance on application authentication is generally focused on applications connecting to web and/or application servers (i.e. multi-tier), and a backend database. For the approach with Android mobile apps, we need to recognize that some apps may have multi-tier implementations, whereas others may be confined to on-device installation. Either way, the premise of identifying all potential authentication points is crucial to providing proper access control.

[[4]](#footnote-4)

Figure 3‑1 Potential Sample Authentication

The sample application described from DISA guidance identifies a multi-tier application that connects to middleware servers and backend databases. If such an Android application is being developed, the developers shall review section 3.8 in “Application Security and Development STIG, V3R2” and recognize the necessity to substitute some assumptions. A model as defined above will require DoD Public Key Infrastructure (PKI)-approved credentials as detailed in DoDI 8520.2 and would require an approved common access card (CAC) reader, either built-in to an Android device or external. From the perspective of the DISA Application Security STIG, the “web browser” is the Android mobile application, and the “web user” is the Android device user.

For on-device applications that do not interface with remote services or backend databases, the complex authentication model described above does not directly apply to the Android application development model. An Android APK contains the application client, code, data, and database in a single file. “If application users are required to be authenticated, then this authentication must be performed using DoD-approved PKI credentials. Certain applications providing read-only public releasable information may not require user authentication.[[5]](#footnote-5)”

Standalone applications, defined as not providing, using, or otherwise interacting with the network or application services, can make use of other authentication methods. For example, Android OS authentication can be used to establish identity to the application and resources. [[6]](#footnote-6) Android OS authentication is not addressed in this publication.

For additional detailed direction on application authentication, refer to Application Security and Development STIG, V3R2, 3.8 Authentication.

## Password Guidance

PK-enabled applications using public key certifications for authentication are not required to implement password authentication, enforce password complexity, or maintenance requirements. Systems not enabling PKI must meet minimum password requirements, as defined in DoDI 8520.2. In short, DoDI 8520.2 identifies when password-based authentication is allowed; however password-based authentication is highly discouraged.[[7]](#footnote-7) Upon a proven CAC/PIV smart card solution for Android, it is anticipated that Password Guidance will be deprecated for a CAC/PIV solution. If Android devices incorporate hardware cryptographic modules in the future, that also may allow more flexible authentication guidance.

### Password Masks

Passwords shall be masked, so that they are not visibly displayed. Characters, such as asterisks can be displayed instead of the actual password.

### Password Strength Compliance

All account passwords, both administrative and non-administrative, must comply with the following:

* Passwords must be at least 8 characters long.[[8]](#footnote-8)
* Passwords must contain a mix of upper case letters, lower case letters, and numbers.
* When a password is changed, users must not be able to use personal information such as names, telephone numbers, account names, or dictionary words.
* Passwords must expire after 60 days.
* Users must not be able to reuse any of their previous 10 passwords.
* Users must not be able to change passwords more than once a day, except in the case of an administrator or a privileged user. Privileged users may be required to reset a user’s forgotten passwords and the ability to change passwords more than once per day.
* When a password is changed, the new password must differ from the previous password by at least four characters.

If applications transmit account passwords, they must be transmitted in an encrypted format. Unclassified systems must adhere to FIPS 140-2 encryption module standards. Classified systems require NSA/COMSEC-approved Type-1 encryption or approval through another NSA process such as the emerging NSA Commercial Solutions for Classified process.

### Notice and Consent Banner

Android (and all Personal Electronic Devices) require the use of a standard Notice and Consent Banner. Deviations to the standard banner are not permitted except as authorized in writing by the Deputy Assistant Secretary of Defense for Information and Identity Assurance. The banner shall be implemented as a click-through (or modal) at logon, blocking further access into the application until the user consents, or taps “OK”, to the banner. The banner text should be customizable in the event of future updates.

For Android applications, use the following banner:

[[9]](#footnote-9)

Figure 3‑2 Standard DoD PED Banner

In certain environments, the Notice and Consent Banner may have adverse operational security impacts by advertising the fact that the device is associated with the U.S. Government. We recommend that exceptions should be granted from this requirement when appropriate.

# Data Protection

Data from Android applications are typically stored in local SQLite databases, located in the application’s directory within the /data partition. The database stores all data in plain text, allowing easy ability for extraction of the .db file and viewing contents in a relational database management system (RDBMS) or other tool. Applications have the ability to store data in other formats or in other directories.

This section will focus on options to store sensitive information within an Android application.

## Database Encryption

### Database Encryption Guidance

Sensitive and personally identifiable information (PII), as defined by Application Security and Development STIG V3R2 and FIPS PUB 140-2, shall be protected by means of FIPS 140-2 compliant cryptographic algorithms. If cryptographic technologies are not implemented correctly, it may be possible for an attacker to access protected data.

### Potential Implementations

A challenge for Android applications is that there is no known FIPS 140-2 compliant cryptographic module for SQLite databases. There are, however, some solutions that claim to provide the level of encryption that NIST requires for FIPS 140-2 testing.

Three examples of database encryption include SQLite Encryption Extension (SEE)[[10]](#footnote-10), SQLite Crypt[[11]](#footnote-11), and SQLCipher[[12]](#footnote-12). These are add-ons to SQLite which allows an application to read and write encrypted database files using the Advanced Encryption Standard (AES), and the add-ons could be vetted by NIST for FIPS 140-2 compliancy. This level of protection for sensitive data is necessary in order to assure data protection for an extended period of time.

Known implementations of encrypted Android SQLite databases, are utilizing wxSQLite. wxSQLite[[13]](#footnote-13) includes a SQLite compatible encryption codec in C using the Android Native Development Kit (NDK), and leverages AES. When implemented it is transparent to the developer and user, though it relies on a key to access the database. NOTE: This method is merely a deterrent, as a persistent attacker could locate the on-device key to decrypt the database.

## SD Card Storage

Applications should avoid storing data on an external SD card or in the /sdcard partition. It is common for developers to store large databases (i.e. >100MB) on SD cards to facilitate the need for extra storage, but this presents security risks. The /sdcard partition within Android uses a file system that does not support file ownership or file permissions. Therefore, files stored in the /sdcard partition can be accessed or modified by other Android applications. Also, it is trivial for someone with physical access to the Android device to remove the SD card and retrieve or modify its contents, while the internal storage device is more difficult for an attacker to gain physical access to.

### SD Card Storage Guidance

Any use of SD card storage by an application shall be documented as a necessity. The certifying authority must approve data and/or application storage via SD card before it will be approved for usage. Applications must employ encryption and integrity protection on data stored on the SD card, unless it can be demonstrated that the data is not sensitive in any way and that malicious or accidental modifications to the data will have no adverse impact on the application.

## Android Application Package

Application developers need to recognize that anyone with access to an Android device can trivially retrieve the .apk application packages from the device, then decompile or reverse engineer the application. Sensitive information such as encryption keys should not be stored within the application code.

## File Permissions

Android file systems (except the /sdcard partition) support UNIX file permissions. Each application owns its own directory on the /data partition and has the ability to set its own file permissions on its files and directories.

### File Permission Guidance

Each application’s files and directories should not be set world-readable or world-writable. World-readable files can be accessed by other applications. World-writable files can be modified by other applications. Regardless of read permission, data stored on the device should be encrypted, as discussed in section 3.1, to prevent data leakage in the event of the device being compromised.

#

# Follow Secure Programming Practices

Existing Java and Android secure coding guidelines should be leveraged whenever possible. Oracle (formerly Sun) provides a "Secure Coding Guidelines for the Java Programming Language" document[[14]](#footnote-14). iSEC Partners provides a "Developing Secure Mobile Applications for Android" paper[[15]](#footnote-15). In this section, we describe important secure programming practices, but this section is not all-inclusive. The other sections of this document also describe secure programming practices applicable to those specific sections.

## Input Validation

Text fields can potentially be influenced by an attacker to contain characters that maliciously impact software components.

### Input Validation Guidance

Whenever possible, text fields should be strongly typed, meaning that the allowed characters, lengths, and ranges are known and enforced through input validation. Whitelisting (determining and enforcing the allowed content) is generally a better approach than blacklisting (removing specifically disallowed characters), as it is difficult to anticipate all potentially malicious content.

## Avoiding SQL injection attacks

Android applications often use SQLite databases to store and retrieve data. SQL injection vulnerabilities, commonly found in web applications, can occur in Android applications as well. In an SQL injection, an attacker maliciously inputs SQL commands through a data input field, causing unauthorized modifications to or information leakage from an SQL database.

### Input via SQL Guidance

To avoid SQL injection vulnerabilities, user input should not be directly passed to SQL queries (for instance, through string concatenation). Instead, prepared or parameterized statements should be used, which ensure that the inputs to the SQL database are properly sanitized and do not contain malicious content. Android's SQLiteDatabase provides convenience methods to delete, insert, replace, and update database rows. These methods make use of SQL prepared statements and avoid SQL injection vulnerabilities. We recommend these methods be used when possible.

In some cases, the convenience methods may not be an option. In those cases, any time a dynamic input value is used (an input whose value could be set or influenced with malicious intent), a parameterized query needs to be used to avoid SQL injection vulnerabilities. An example of an SQL parameterized query:

CORRECT: db.rawQuery("SELECT \* from table where id = ? and name = ?", new String[] { id, name});

INCORRECT: db.rawQuery("SELECT \* from table where id =" + id + " and name =" + name, null); This code segment is potentially vulnerable because an attacker could inject malicious SQL statements into the id or name variables.

## Avoiding command injection attacks

Command injection vulnerabilities are possible in Android applications. In a command injection attack, the attacker injects malicious commands into the variables passed to certain functions, which results in the malicious commands being executed. The DISA Application Security and Development STIG lists methods in several programming languages that are susceptible to command injection attacks. The Java methods listed are: Class.forName(), class.newInstance(), and Runtime.exec(). Use of these methods should be avoided. There are often safer Java alternatives to calling these methods. For example, perform filesystem actions using the java.io.File classes rather than invoking Linux shell commands through Runtime.exec(). However, when the use of these methods is necessary, avoid dynamic inputs to these methods that can be influenced by a malicious entity. Instead, use a statically defined list of the set of allowed inputs. If the inputs can be set or influenced externally, then a whitelist of allowed characters, as discussed above, should be used. For instance, only allowing alphanumeric values and blocking all other characters.

## Sign Application Packages

All Android packages must be signed by a private key associated with a certificate held by the developer or other appropriate authority (such as an application store certificate, whose presence indicates that the application has been vetted for use).

The digital signature provides integrity protection over the application, as any unauthorized modifications will be detected. The digital signature enables Android's signature-based permissions enforcement, allowing sharing between applications that are signed by the same entity. Application upgrades are also protected by digital signatures and certificates by ensuring that the new version of the same application has been signed by the same key.

The stock Android operating system and application installer currently do not use digital signatures and certificates to guarantee authenticity. Applications signed by any entity, even a completely unknown entity, can be installed. Malicious applications, or applications from unknown or less-trusted developers whose security properties are unknown, are a huge potential attack vector.

In the future, whether the feature is added by Google or another party, the Android operating system ideally would provide the ability to only allow applications that are digitally signed by a trusted entity to be installed or executed. Requiring that all Android packages be signed by a private key associated with a trusted certificate holder helps prepare for that future ability, and also provides the ability now for an enterprise application market to automatically verify application authenticity or for other out-of-band verification of application authenticity.

### Signed Code Identification

All signed code must be signed with a DoD PKI mobile code signing certificate. Signed code certificates must be validated as indicated in the PKI Certificate Validation section before they are executed on a workstation.[[16]](#footnote-16)

## Avoid Android NDK or Java JNI Use, Unless Necessary

The Android NDK and Java JNI provide the ability to implement portions of applications using native code written in languages such as C and C++.

The Android application permissions scheme and “sandbox” protections still apply to native code. However, many of the discovered Android privilege escalation exploits require native code execution ability in order to exploit Linux kernel vulnerabilities or vulnerabilities in Android OS system processes. Also, code analysis tools used to discover vulnerabilities or malicious behavior in Android applications are more likely to be able to examine Java code rather than native code. DARPA’s BAA-11-63 effort to perform automated analysis of Android applications is specifically excluding the ability to analyze native code at this time. Also, using the NDK eliminates many of the security protections provided by the Java language such as protection against buffer overflows.

According to Google's Android developer guidance[[17]](#footnote-17), the "NDK will not benefit most applications… using native code does not result in an automatic performance increase, but always increases application complexity. In general, you should only use native code if it is essential to your application, not just because you prefer to program in C/C++."

Due to the increased complexity and security risks involved with native code, application developers should avoid use of the Android NDK or Java JNI unless their use is necessary. Developers should justify in writing any use of the NDK or JNI, and these uses should be scrutinized as part of a certification and accreditation process.

## Third-Party Libraries

In some cases, code may already exist that provides desired functionality. However, caution should be used when relying on third-party libraries. Libraries from unknown or untrusted sources should not be used. Such libraries could contain code that performs more than the desired actions. Utilizing an external library allows it to act with the permissions of the developed application. Malicious third-party libraries could probe for permissions or transmit sensitive phone information.

#

# Secure Data Communication

Mobile devices typically rely on cellular or Wi-Fi networks for connectivity. Cellular networks frequently use weak cryptography that can be compromised by adversaries to eavesdrop on or manipulate network traffic. Cellular networks may be controlled by potentially untrusted third parties or may have been infiltrated by unknown parties. Wi-Fi networks may also be controlled by potentially untrusted third parties or utilized by malicious users. Routing of Internet communication is unpredictable. The network path between the endpoints may take data packets through entities with malicious intentions. For all of these reasons, mobile applications must employ end-to-end data in transit protection of any potentially sensitive information, including session cookies or authentication tokens. To be safe, mobile applications should use data-in-transit encryption for all transmitted information unless a specific reason exists not to. Transport Layer Security (TLS), described below, is the typical method for providing this protection. IPsec Virtual Private Networks (VPNs) are another option in some cases.

## Leverage TLS/SSL

Transport Layer Security (TLS), formerly known as Secure Sockets Layer, or SSL) is an Internet Engineering Task Force (IETF) standard for securing IP communications by providing confidentiality and integrity protections to data as well as authentication. TLS is used to secure a variety of protocols including HTTPS. TLS is readily available to Android developers. This section provides guidance on utilizing TLS within Android applications.

### Use of Appropriate TLS Cipher Suites and TLS Protocol Version

TLS clients and servers negotiate a cipher suite to use. A cipher suite is a set of cryptographic algorithms used to protect the TLS session. Cipher suite negotiation provides the flexibility to adapt to new cryptographic algorithms over time. TLS supports a wide variety of cryptographic algorithms and cipher suites. Android's default TLS implementation has many cipher suites enabled by default, and a number of these are insecure choices or are otherwise not approved for U.S. Government use. NIST Special Publication 800-57 Part 3 provides guidance on appropriate U.S. Government TLS cipher suite choices. If NSA Suite B compliance is necessary, IETF RFC 5430 (soon to be updated by IETF RFC 5430bis) provides stricter guidance.

Multiple versions of the TLS/SSL protocol exist, but some of the older versions are no longer considered secure and must not be used. SSL version 2.0 must not be used under any circumstance as it is insecure. SSL version 3.0 is not approved for federal government use according to NIST Special Publication 800-52. TLS versions 1.0, 1.1, 1.2, or above are generally acceptable for use, although Suite B compliance requires TLS version 1.2 or above. Android's default crypto library appears to acceptably use SSL version 3.0 and TLS version 1.0 by default.

In environments using RSA PKI certificates, we recommend enabling these widely supported cipher suites that are US Government approved, listed in this priority order (most preferred first):

TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA

The cipher suites containing "DHE" as part of the name perform an ephemeral Diffie-Hellman key exchange that provides additional protection against decrypting the sessions, but those cipher suites may not be supported by all servers.

If Suite B ECDSA PKI certificates are being used, the TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA and TLS\_ECDHE\_ECSDA\_WITH\_AES\_128\_CBC\_SHA cipher suites are supported by newer versions of the Android OS. These cipher suites are not fully Suite B compliant but can be used for a transitional period until the fully compliant cipher suites are available (TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384 and TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256).

In Java, the SSLSocket or SSLServerSocket class's sslEnabledCipherSuites method and the SSLParameters class's setCipherSuites method provide the ability to specify the allowable cipher suites.

### Certificate Validation

TLS clients authenticate the server's identity using the server's X.509 PKI certificate. The server may optionally authenticate the client's identity using a client X.509 PKI certificate. PKI certificate validation relies on a list of root certification authorities (CAs) that are trusted to issue certificates. Certificates are only valid if a certificate trust chain can be constructed between the certificate and one of the trusted root CAs.

Android devices are typically distributed with over 100 root CA certificates from a variety of certification authorities all around the world. These certification authorities all have equal ability to create certificates that will be trusted by default to assert identities. If any of these CAs are controlled or compromised by a malicious adversary, the adversary could easily construct falsified certificates. Most of these CAs are not needed and just increase the attack surface against devices. Also, the root certificate list generally does not include the DoD PKI root CA or other US Government root CAs. The root certificate list is stored in the read-only /system partition where it is difficult to customize or manage at an enterprise level (modification requires root access to the device).

Typical Android applications only need to trust the specific root CAs used to sign certificates for the specific servers contacted by the applications. For example, the DoD PKI root CAs are used to issue certificates to DoD NIPRNET and Internet web servers. Android applications only communicating with these servers only need to trust the DoD PKI root CA certificates and not other root CA certificates.

Below is a code example of initiating an HTTPS connection to [https://www.example.com/](http://developer.android.com/reference/android/Manifest.permission.html) with a specific list of trusted root certificates. The file keystore.bks is a Bouncy Castle-formatted key store containing the trusted certificates:[[18]](#footnote-18)

KeyStore keyStore = KeyStore.getInstance("BKS");

FileInputStream fis = null;

try {

 fis = new FileInputStream("keystore.bks");

 keyStore.load(fis, null);

} finally {

 if (fis != null) {

 fis.close();

 }

}

TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");

tmf.init(keyStore);

SSLContext context = SSLContext.getInstance("TLS");

Context.init(null, tmf.getTrustManagers(), null);

URL url = new URL("[https://www.example.com/](http://developer.android.com/reference/android/Manifest.permission.html)");

HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();

urlConnection.setSSLSocketFactory(context.getSocketFactory());

inputStream in = urlConnection.getInputStream();

### FIPS Validated Implementations

FIPS validated cryptographic implementations are required for protection of sensitive U.S. Government information. Android, by default, uses a combination of the BouncyCastle and OpenSSL cryptographic libraries, which are not known to be FIPS validated. Mocana and RSA BSAFE are two examples of FIPS validated cryptographic libraries with Java support. To use a FIPS validated cryptographic library for cryptographic operations, it needs to be added to the application's crypto provider list. The specific FIPS cryptographic library should contain the instructions for adding it as a crypto provider. A future version of this document may provide more details on how to add a FIPS validated cryptographic library.

### Remote Authentication Best Practices

In typical commercial practice, TLS/SSL only authenticates the server's PKI certificate, because clients often do not have certificates. Then, once the session is established, the client authenticates using another technique. However, we recommend that clients should be issued certificates and use those for TLS/SSL client certificate authentication (mutual authentication using both server and client certificates), as this is a strong authentication method.

If client certificates are not available, username and password authentication is typically performed. Storing passwords on the Android device is prohibited, as that places the password at increased risk of compromise. Instead, the password should be passed to the remote server to obtain an authentication token or other long-term session identifier, the password should be cleared from the Android device's memory, and the authentication token should be stored on the Android device for future use of the remote server. If the authentication token is compromised, it can be revoked by the remote server, and unlike a password, the token is not potentially re-used on multiple servers, avoiding placing other servers that use the same password at risk.

### Virtual Private Networks

Virtual Private Networks (VPNs) provide the ability to establish an encrypted tunnel from the mobile device through an untrusted network (such as a cellular network, Wi-Fi network, or the Internet) to a trusted network enclave, effectively placing the mobile device onto that trusted network enclave.

IPsec is the industry standard typically used to establish VPNs. NIST Special Publication 800-77 and 800-57 Part 3 provide guidance for acceptable U.S. Government use of IPsec.

Android includes IPsec VPN support. However, the default VPN client built in to typical Android devices is not FIPS validated. FIPS validated IPsec VPN clients should be preferred and are required for connectivity to government networks.

## Parameter Content

Even when using end-to-end encryption, it is best not to abuse sensitive information. Sensitive phone information such as phone number, serial number, and device ID should not be used to identify connections[[19]](#footnote-19). It is not uncommon for information such as this to be used as HTTP parameters or as device fingerprints because it provides quick access to a unique number. However, this practice is discouraged as it makes more personally identifiable information available to be captured than necessary.

# Secure Inter-App Communication

Android provides mechanisms that enable communication between applications. These mechanisms have great utility but also introduce security risks. This section describes how to secure inter-application communication. The sources for this information include Chin, et al.’s “Analyzing Inter-Application Communication in Android” paper[[20]](#footnote-20), the iSEC Partners document titled “A Study of Android Application Security”[[21]](#footnote-21), and the iSEC Partners white paper “Developing Secure Mobile Applications for Android”[[22]](#footnote-22).

## Securing Android Intents

Intents are an Android OS mechanism used for both intra-application and inter-application communication. Explicit Intents specify a particular component to be delivered to, while implicit Intents are delivered by the system to any of the components that declares support for the requested operation. We describe the potential risks that apply to Intent senders and to Intent receivers along with suggested mitigations.

### Sender Risks and Mitigations

This section describes the risks that apply to Intent senders along with suggested mitigations.

#### Implicit Intents

Implicit Intents can potentially be intercepted by malicious applications. A malicious application could receive implicit Intents by declaring wide-ranging Intent filters. The malicious application would gain access to the data in the Intent, and the malicious application could potentially be able to conduct other attacks. For example, a malicious application could conduct a phishing attack in the case of an Intent that signals activation of a password prompt or credit card prompt display. Also, when Intents are used to signal an Activity or Service, the Activity or Service has the ability to return information to the sender. If a malicious Activity or Service is inadvertently called, the responses could be malicious and be used to trigger attacks against the sender if not handled properly.

To mitigate these threats, applications should use explicit Intents when it is possible to address Intents to a specific component, and must use explicit Intents if the Intent is for an internal component (intra-application communication).

When sending broadcast Intents, applications can define custom permissions and only send the Intent to receivers that hold the permission. Permissions are an effective security mechanism when they are combined with an effective enterprise management approach that provides control over the permissions held by other installed applications to ensure that they do not maliciously request the permission. For additional protection, the custom permission could be declared as a "signature" level permission, which would require both applications be signed by the same private key (usually meaning they are both written by the same developer).

The same permissions approach is not available for sending implicit Intents to an Activity or Service. Explicit Intents should be used when possible, but if an implicit Intent is necessary, applications should avoid placing sensitive information in implicit Intents sent to an Activity or Service and should perform input validation on any received responses.

#### Explicit Intents

Explicit Intents are safer than implicit Intents because they are addressed to a specific component. Explicit Intents should be used when possible but may not always be appropriate because of their lack of flexibility. Explicit Intents still present some risks. A malicious application could be installed that takes the name of another intended component. A potential mitigation against this risk is enterprise management control over installed applications. Also, before sending an explicit Intent to another component, an application can verify that the other component has a required permission. Taken from iSEC Partners guidance[[23]](#footnote-23), this code verifies that the other package has the required permission:

res = getPackageManager().checkPermission(permToCheck, name.getPackageName());

then compare res with PackageManager.PERMISSION\_GRANTED or PackageManager.PERMISSION\_DENIED

As described in the above section, an enterprise management approach is needed to control the permissions held by other installed applications to ensure that they do not maliciously request the permission.

Another, stronger approach is to obtain the receiving application's package signature and ensure that it matches an expected value.

### Receiver Risks and Mitigations

This section describes the risks that apply to Intent receivers along with suggested mitigations. Intent receivers could receive unanticipated Intents from malicious sender applications, known as spoofed Intents.

Many application components are only used internally and have no need to be invoked by external applications. To be safe, developers should declare these components in the manifest as explicitly internal:

<activity android:name=".TestActivity"

android:exported="false">

</activity>[[24]](#footnote-24)

As described in Enck, et al., applications must perform null checks on all received objects. Otherwise, an unintended dereference of a null variable, potentially provided by a malicious application through an Intent, may cause the application to crash.

Broadcast Receivers are at risk of attack. As described by Chin, et al., certain Intents are supposed to be broadcast by only the operating system (rather than by any application), so these Intents may be given more trust by the application. However, when applications declare the ability to receive broadcast messages from the system, they are able to receive messages from any other application, not just the system. Applications must be sure to explicitly ensure that the received message matches a system action (which can only be sent by the operating system) before proceeding:

public void onReceive(Context ctxt, Intent i){

if (i.getAction().equals("expected.action"))

return;

}[[25]](#footnote-25)

Intent filters are not a security mechanism. Even if an Intent recipient (such as an Activity, Service, or BroadcastReceiver) declares an IntentFilter, a malicious sender can still send an explicit Intent to invoke the activity with an Intent that does not match the filter. Recipients must be prepared to receive unexpected data in Intents and appropriately filter their contents.

Recipients can use permissions to control who can invoke them. These permissions are checked by the Activity Manager to ensure the sender meets the requirements to send the Intent before delivering it.

For example, the following activity can only be invoked by applications that hold the custom permission "my.permission":

<activity android:name=".TestActivity2"

android:exported="true">

android:permission="my.permission">

<intent-filter>

<action android:name="my.action.TEST"/>

</intent-filter>

</activity>[[26]](#footnote-26)

Permissions are an effective strategy when combined with an enterprise management ability to control the permissions held by installed applications. A stronger approach is for the invoked application to use the PackageManager class to obtain the sending application's package name and package signature and ensure that they both match an expected value.

### Sticky Intent

Sticky Intents are sent and received through sticky Broadcasts. They are intended to share information about the system state[[27]](#footnote-27). Applications cannot limit access to sticky Intents based on permissions. The result is that any application with the BROADCAST\_STICKY permission has full access to all sticky Intents. Therefore, sticky Intents should not be used. Sensitive data could be easily captured y by any listening application. A malicious application could also simply remove the Intent, breaking down the communication channel.

### Intent Reflection

When an application sends an Intent in response to receiving an Intent, it is at risk of intent reflection. Intent reflection occurs when an application has another application send an Intent on its behalf[[28]](#footnote-28). An application can use this to send Intents it may not otherwise be able to send, which is a form of privilege escalation. This is avoided by accepting a PendingIntent instead of an Intent. PendingIntents are sent as the creating process, and are thus rejected if the process does not have permission to send the Intent. When sending PendingIntents, it is important to trust the application the PendingIntent is sent to, as the application will be sending the Intent on the caller's behalf.

## Securing Content Providers

Android's ContentProvider mechanism can be used to share data with other applications or internally with the same application. Content Providers must only be exported if other applications have a need to use them. If exported, the application developer must define read and write permissions to prevent the ContentProvider from being misused by malicious applications. Just as described earlier, permissions are only effective if combined with an effective enterprise management strategy to ensure that each application only requests and obtains appropriate permissions.

#

# Application Update Process

Although not detailed in this guide, a process for application maintenance and sustainment is critical for complete application lifecycle. A formal change management process should be documented based on program expectations on maintenance and sustainment. Two important areas to be included in the application update process are:

* Creation of secure/strong process for updating apps.
* Ability to rapidly updates/patches which adhere to CVE (or other identified vulnerabilities).

# Non-Android SDK Applications

## Browser-based Apps

Android's permissions scheme is implemented at the kernel level. Any browser-based apps will only be able to do what the browser lets them do; and in turn, the browser will be beholden to the usual permissions scheme. Therefore, browser-based apps can safely be treated with the same rigor than native apps.

## Adobe Air Apps

Adobe’s Flash technology now allows mobile application development for cross-platform usage, including Android. Air/Flex/Flash does not allow for access to Manifest permissions, which is restricting of device permissions an application is granted. We are assessing the Air/Flex/Flash permission model to determine if the Adobe application development will be viable for DoD mobile apps.

At this writing, Adobe Air applications should not be deemed acceptable offerings in the Android app store.

This page intentionally left blank.

###### Android Manifest Permissions

This list is derived from SDK documentation at the Android Developers portal.

|  |
| --- |
| Constants |
| [ACCESS\_CHECKIN\_PROPERTIES](http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_CHECKIN_PROPERTIES) | Allows read/write access to the "properties" table in the checkin database, to change values that get uploaded. |
| [ACCESS\_COARSE\_LOCATION](http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_COARSE_LOCATION) | Allows an application to access coarse (e.g., Cell-ID, WiFi) location  |
| [ACCESS\_FINE\_LOCATION](http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_FINE_LOCATION) | Allows an application to access fine (e.g., GPS) location  |
| [ACCESS\_LOCATION\_EXTRA\_COMMANDS](http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_LOCATION_EXTRA_COMMANDS) | Allows an application to access extra location provider commands  |
| [ACCESS\_MOCK\_LOCATION](http://developer.android.com/reference/android/inputmethodservice/InputMethodService.html#ACCESS_MOCK_LOCATION) | Allows an application to create mock location providers for testing  |
| [ACCESS\_NETWORK\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_NETWORK_STATE) | Allows applications to access information about networks  |
| [ACCESS\_SURFACE\_FLINGER](http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf#ACCESS_SURFACE_FLINGER) | Allows an application to use SurfaceFlinger's low level features  |
| [ACCESS\_WIFI\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#ACCESS_WIFI_STATE) | Allows applications to access information about Wi-Fi networks  |
| [ACCOUNT\_MANAGER](http://developer.android.com/reference/android/Manifest.permission.html#ACCOUNT_MANAGER) | Allows applications to call into AccountAuthenticators. |
| [AUTHENTICATE\_ACCOUNTS](http://developer.android.com/reference/android/Manifest.permission.html#AUTHENTICATE_ACCOUNTS) | Allows an application to act as an AccountAuthenticator for the AccountManager  |
| [BATTERY\_STATS](http://developer.android.com/reference/android/Manifest.permission.html#BATTERY_STATS) | Allows an application to collect battery statistics  |
| [BIND\_APPWIDGET](http://developer.android.com/reference/android/Manifest.permission.html#BIND_APPWIDGET) | Allows an application to tell the AppWidget service which application can access AppWidget's data. |
| [BIND\_DEVICE\_ADMIN](http://developer.android.com/reference/android/Manifest.permission.html#BIND_DEVICE_ADMIN) | Must be required by device administration receiver, to ensure that only the system can interact with it. |
| [BIND\_INPUT\_METHOD](http://developer.android.com/reference/android/Manifest.permission.html#BIND_INPUT_METHOD) | Must be required by an [InputMethodService](http://developer.android.com/reference/android/Manifest.permission.html), to ensure that only the system can bind to it. |
| [BIND\_REMOTEVIEWS](http://developer.android.com/reference/android/Manifest.permission.html#BIND_REMOTEVIEWS) | Must be required by a [RemoteViewsService](http://developer.android.com/reference/android/Manifest.permission.html), to ensure that only the system can bind to it. |
| [BIND\_WALLPAPER](http://code.google.com/p/android/issues/detail#BIND_WALLPAPER) | Must be required by a [WallpaperService](http://developer.android.com/reference/android/Manifest.permission.html), to ensure that only the system can bind to it. |
| [BLUETOOTH](http://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH) | Allows applications to connect to paired bluetooth devices  |
| [BLUETOOTH\_ADMIN](http://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH_ADMIN) | Allows applications to discover and pair bluetooth devices  |
| [BRICK](http://developer.android.com/reference/android/Manifest.permission.html#BRICK) | Required to be able to disable the device (very dangerous!). |
| [BROADCAST\_PACKAGE\_REMOVED](http://developer.android.com/reference/android/Manifest.permission.html#BROADCAST_PACKAGE_REMOVED) | Allows an application to broadcast a notification that an application package has been removed. |
| [BROADCAST\_SMS](http://developer.android.com/reference/android/Manifest.permission.html#BROADCAST_SMS) | Allows an application to broadcast an SMS receipt notification  |
| [BROADCAST\_STICKY](http://developer.android.com/reference/android/Manifest.permission.html#BROADCAST_STICKY) | Allows an application to broadcast sticky intents. |
| [BROADCAST\_WAP\_PUSH](http://developer.android.com/reference/android/Manifest.permission.html#BROADCAST_WAP_PUSH) | Allows an application to broadcast a WAP PUSH receipt notification  |
| [CALL\_PHONE](http://developer.android.com/reference/android/Manifest.permission.html#CALL_PHONE) | Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed. |
| [CALL\_PRIVILEGED](http://www.cs.berkeley.edu/~afelt/intentsecurity-mobisys.pdf#CALL_PRIVILEGED) | Allows an application to call any phone number, including emergency numbers, without going through the Dialer user interface for the user to confirm the call being placed. |
| [CAMERA](http://developer.android.com/reference/android/Manifest.permission.html#CAMERA) | Required to be able to access the camera device. |
| [CHANGE\_COMPONENT\_ENABLED\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#CHANGE_COMPONENT_ENABLED_STATE) | Allows an application to change whether an application component (other than its own) is enabled or not. |
| [CHANGE\_CONFIGURATION](http://developer.android.com/reference/android/Manifest.permission.html#CHANGE_CONFIGURATION) | Allows an application to modify the current configuration, such as locale. |
| [CHANGE\_NETWORK\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#CHANGE_NETWORK_STATE) | Allows applications to change network connectivity state  |
| [CHANGE\_WIFI\_MULTICAST\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#CHANGE_WIFI_MULTICAST_STATE) | Allows applications to enter Wi-Fi Multicast mode  |
| [CHANGE\_WIFI\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#CHANGE_WIFI_STATE) | Allows applications to change Wi-Fi connectivity state  |
| [CLEAR\_APP\_CACHE](http://developer.android.com/reference/android/Manifest.permission.html#CLEAR_APP_CACHE) | Allows an application to clear the caches of all installed applications on the device. |
| [CLEAR\_APP\_USER\_DATA](http://developer.android.com/reference/android/Manifest.permission.html#CLEAR_APP_USER_DATA) | Allows an application to clear user data  |
| [CONTROL\_LOCATION\_UPDATES](http://developer.android.com/reference/android/Manifest.permission.html#CONTROL_LOCATION_UPDATES) | Allows enabling/disabling location update notifications from the radio. |
| [DELETE\_CACHE\_FILES](http://developer.android.com/reference/android/Manifest.permission.html#DELETE_CACHE_FILES) | Allows an application to delete cache files. |
| [DELETE\_PACKAGES](http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf#DELETE_PACKAGES) | Allows an application to delete packages. |
| [DEVICE\_POWER](http://developer.android.com/reference/android/Manifest.permission.html#DEVICE_POWER) | Allows low-level access to power management  |
| [DIAGNOSTIC](http://developer.android.com/reference/android/Manifest.permission.html#DIAGNOSTIC) | Allows applications to RW to diagnostic resources. |
| [DISABLE\_KEYGUARD](http://developer.android.com/reference/android/Manifest.permission.html#DISABLE_KEYGUARD) | Allows applications to disable the keyguard  |
| [DUMP](http://developer.android.com/reference/android/Manifest.permission.html#DUMP) | Allows an application to retrieve state dump information from system services. |
| [EXPAND\_STATUS\_BAR](http://developer.android.com/reference/android/Manifest.permission.html#EXPAND_STATUS_BAR) | Allows an application to expand or collapse the status bar. |
| [FACTORY\_TEST](http://developer.android.com/reference/android/Manifest.permission.html#FACTORY_TEST) | Run as a manufacturer test application, running as the root user. |
| [FLASHLIGHT](http://developer.android.com/reference/android/Manifest.permission.html#FLASHLIGHT) | Allows access to the flashlight  |
| [FORCE\_BACK](http://developer.android.com/reference/android/Manifest.permission.html#FORCE_BACK) | Allows an application to force a BACK operation on whatever is the top activity. |
| [GET\_ACCOUNTS](http://developer.android.com/reference/android/Manifest.permission.html#GET_ACCOUNTS) | Allows access to the list of accounts in the Accounts Service  |
| [GET\_PACKAGE\_SIZE](http://www.oracle.com/technetwork/java/seccodeguide-139067.html#GET_PACKAGE_SIZE) | Allows an application to find out the space used by any package. |
| [GET\_TASKS](http://developer.android.com/reference/android/Manifest.permission.html#GET_TASKS) | Allows an application to get information about the currently or recently running tasks: a thumbnail representation of the tasks, what activities are running in it, etc. |
| [GLOBAL\_SEARCH](http://developer.android.com/reference/android/service/wallpaper/WallpaperService.html#GLOBAL_SEARCH) | This permission can be used on content providers to allow the global search system to access their data. |
| [HARDWARE\_TEST](http://developer.android.com/reference/android/Manifest.permission.html#HARDWARE_TEST) | Allows access to hardware peripherals. |
| [INJECT\_EVENTS](http://developer.android.com/reference/android/Manifest.permission.html#INJECT_EVENTS) | Allows an application to inject user events (keys, touch, trackball) into the event stream and deliver them to ANY window. |
| [INSTALL\_LOCATION\_PROVIDER](http://developer.android.com/reference/android/widget/RemoteViewsService.html#INSTALL_LOCATION_PROVIDER) | Allows an application to install a location provider into the Location Manager  |
| [INSTALL\_PACKAGES](http://developer.android.com/reference/android/Manifest.permission.html#INSTALL_PACKAGES) | Allows an application to install packages. |
| [INTERNAL\_SYSTEM\_WINDOW](http://developer.android.com/reference/android/Manifest.permission.html#INTERNAL_SYSTEM_WINDOW) | Allows an application to open windows that are for use by parts of the system user interface. |
| [INTERNET](https://www.example.com/#INTERNET) | Allows applications to open network sockets. |
| [KILL\_BACKGROUND\_PROCESSES](http://developer.android.com/reference/android/Manifest.permission.html#KILL_BACKGROUND_PROCESSES) | Allows an application to call [killBackgroundProcesses(String)](http://developer.android.com/reference/android/Manifest.permission.html#killBackgroundProcesses%28java.lang.String%29). |
| [MANAGE\_ACCOUNTS](http://developer.android.com/reference/android/content/Intent.html#MANAGE_ACCOUNTS) | Allows an application to manage the list of accounts in the AccountManager  |
| [MANAGE\_APP\_TOKENS](http://developer.android.com/reference/android/Manifest.permission.html#MANAGE_APP_TOKENS) | Allows an application to manage (create, destroy, Z-order) application tokens in the window manager. |
| [MASTER\_CLEAR](http://developer.android.com/reference/android/Manifest.permission.html#MASTER_CLEAR) |  |
| [MODIFY\_AUDIO\_SETTINGS](http://developer.android.com/reference/android/Manifest.permission.html#MODIFY_AUDIO_SETTINGS) | Allows an application to modify global audio settings  |
| [MODIFY\_PHONE\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#MODIFY_PHONE_STATE) | Allows modification of the telephony state - power on, mmi, etc. |
| [MOUNT\_FORMAT\_FILESYSTEMS](http://developer.android.com/reference/android/Manifest.permission.html#MOUNT_FORMAT_FILESYSTEMS) | Allows formatting file systems for removable storage. |
| [MOUNT\_UNMOUNT\_FILESYSTEMS](http://developer.android.com/reference/android/Manifest.permission.html#MOUNT_UNMOUNT_FILESYSTEMS) | Allows mounting and unmounting file systems for removable storage. |
| [NFC](http://developer.android.com/reference/android/Manifest.permission.html#NFC) | Allows applications to perform I/O operations over NFC  |
| [PERSISTENT\_ACTIVITY](http://developer.android.com/reference/android/Manifest.permission.html#PERSISTENT_ACTIVITY) | This constant is deprecated. This functionality will be removed in the future; please do not use. Allow an application to make its activities persistent.  |
| [PROCESS\_OUTGOING\_CALLS](http://developer.android.com/reference/android/Manifest.permission.html#PROCESS_OUTGOING_CALLS) | Allows an application to monitor, modify, or abort outgoing calls. |
| [READ\_CALENDAR](http://developer.android.com/reference/android/Manifest.permission.html#READ_CALENDAR) | Allows an application to read the user's calendar data. |
| [READ\_CONTACTS](http://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) | Allows an application to read the user's contacts data. |
| [READ\_FRAME\_BUFFER](http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf#READ_FRAME_BUFFER) | Allows an application to take screen shots and more generally get access to the frame buffer data  |
| [READ\_HISTORY\_BOOKMARKS](http://developer.android.com/reference/android/Manifest.permission.html#READ_HISTORY_BOOKMARKS) | Allows an application to read (but not write) the user's browsing history and bookmarks. |
| [READ\_INPUT\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#READ_INPUT_STATE) | Allows an application to retrieve the current state of keys and switches. |
| [READ\_LOGS](http://developer.android.com/reference/android/Manifest.permission.html#READ_LOGS) | Allows an application to read the low-level system log files. |
| [READ\_PHONE\_STATE](http://developer.android.com/reference/android/Manifest.permission.html#READ_PHONE_STATE) | Allows read only access to phone state. |
| [READ\_SMS](http://developer.android.com/reference/android/Manifest.permission.html#READ_SMS) | Allows an application to read SMS messages. |
| [READ\_SYNC\_SETTINGS](http://developer.android.com/reference/android/content/pm/PackageManager.html#READ_SYNC_SETTINGS) | Allows applications to read the sync settings  |
| [READ\_SYNC\_STATS](http://developer.android.com/reference/android/Manifest.permission.html#READ_SYNC_STATS) | Allows applications to read the sync stats  |
| [REBOOT](http://developer.android.com/reference/android/Manifest.permission.html#REBOOT) | Required to be able to reboot the device. |
| [RECEIVE\_BOOT\_COMPLETED](http://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_BOOT_COMPLETED) | Allows an application to receive the [ACTION\_BOOT\_COMPLETED](http://www.enck.org/pubs/enck-sec11.pdf#ACTION_BOOT_COMPLETED) that is broadcast after the system finishes booting. |
| [RECEIVE\_MMS](http://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_MMS) | Allows an application to monitor incoming MMS messages, to record or perform processing on them. |
| [RECEIVE\_SMS](http://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_SMS) | Allows an application to monitor incoming SMS messages, to record or perform processing on them. |
| [RECEIVE\_WAP\_PUSH](http://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_WAP_PUSH) | Allows an application to monitor incoming WAP push messages. |
| [RECORD\_AUDIO](https://www.example.com/#RECORD_AUDIO) | Allows an application to record audio  |
| [REORDER\_TASKS](http://developer.android.com/reference/android/Manifest.permission.html#REORDER_TASKS) | Allows an application to change the Z-order of tasks  |
| [RESTART\_PACKAGES](http://developer.android.com/reference/android/Manifest.permission.html#RESTART_PACKAGES) | This constant is deprecated. The [*restartPackage(String)*](http://developer.android.com/reference/android/view/WindowManager.LayoutParams.html#restartPackage%28java.lang.String%29) API is no longer supported.  |
| [SEND\_SMS](http://developer.android.com/reference/android/Manifest.permission.html#SEND_SMS) | Allows an application to send SMS messages. |
| [SET\_ACTIVITY\_WATCHER](http://developer.android.com/reference/android/Manifest.permission.html#SET_ACTIVITY_WATCHER) | Allows an application to watch and control how activities are started globally in the system. |
| [SET\_ALARM](http://developer.android.com/reference/android/Manifest.permission.html#SET_ALARM) | Allows an application to broadcast an Intent to set an alarm for the user. |
| [SET\_ALWAYS\_FINISH](http://developer.android.com/sdk/ndk/overview.html#SET_ALWAYS_FINISH) | Allows an application to control whether activities are immediately finished when put in the background. |
| [SET\_ANIMATION\_SCALE](http://developer.android.com/reference/android/Manifest.permission.html#SET_ANIMATION_SCALE) | Modify the global animation scaling factor. |
| [SET\_DEBUG\_APP](http://developer.android.com/reference/android/Manifest.permission.html#SET_DEBUG_APP) | Configure an application for debugging. |
| [SET\_ORIENTATION](http://developer.android.com/reference/android/Manifest.permission.html#SET_ORIENTATION) | Allows low-level access to setting the orientation (actually rotation) of the screen. |
| [SET\_PREFERRED\_APPLICATIONS](http://developer.android.com/reference/android/Manifest.permission.html#SET_PREFERRED_APPLICATIONS) | This constant is deprecated. No longer useful, see [*addPackageToPreferred(String)*](http://developer.android.com/reference/android/Manifest.permission.html#addPackageToPreferred%28java.lang.String%29) for details.  |
| [SET\_PROCESS\_LIMIT](http://developer.android.com/reference/android/Manifest.permission.html#SET_PROCESS_LIMIT) | Allows an application to set the maximum number of (not needed) application processes that can be running. |
| [SET\_TIME](http://developer.android.com/reference/android/Manifest.permission.html#SET_TIME) | Allows applications to set the system time  |
| [SET\_TIME\_ZONE](http://developer.android.com/reference/android/Manifest.permission.html#SET_TIME_ZONE) | Allows applications to set the system time zone  |
| [SET\_WALLPAPER](http://developer.android.com/reference/android/Manifest.permission.html#SET_WALLPAPER) | Allows applications to set the wallpaper  |
| [SET\_WALLPAPER\_HINTS](http://iase.disa.mil/stigs/stig/index.html#SET_WALLPAPER_HINTS) | Allows applications to set the wallpaper hints  |
| [SIGNAL\_PERSISTENT\_PROCESSES](http://developer.android.com/reference/android/Manifest.permission.html#SIGNAL_PERSISTENT_PROCESSES) | Allow an application to request that a signal be sent to all persistent processes  |
| [STATUS\_BAR](http://developer.android.com/reference/android/Manifest.permission.html#STATUS_BAR) | Allows an application to open, close, or disable the status bar and its icons. |
| [SUBSCRIBED\_FEEDS\_READ](http://developer.android.com/reference/android/Manifest.permission.html#SUBSCRIBED_FEEDS_READ) | Allows an application to allow access the subscribed feeds ContentProvider. |
| [SUBSCRIBED\_FEEDS\_WRITE](http://developer.android.com/reference/android/Manifest.permission.html#SUBSCRIBED_FEEDS_WRITE) |  |
| [SYSTEM\_ALERT\_WINDOW](http://developer.android.com/reference/android/Manifest.permission.html#SYSTEM_ALERT_WINDOW) | Allows an application to open windows using the type [TYPE\_SYSTEM\_ALERT](http://developer.android.com/reference/android/Manifest.permission.html#TYPE_SYSTEM_ALERT), shown on top of all other applications. |
| [UPDATE\_DEVICE\_STATS](http://developer.android.com/reference/android/Manifest.permission.html#UPDATE_DEVICE_STATS) | Allows an application to update device statistics. |
| [USE\_CREDENTIALS](http://developer.android.com/reference/android/Manifest.permission.html#USE_CREDENTIALS) | Allows an application to request authtokens from the AccountManager  |
| [USE\_SIP](http://developer.android.com/reference/android/Manifest.permission.html#USE_SIP) | Allows an application to use SIP service  |
| [VIBRATE](http://developer.android.com/reference/android/Manifest.permission.html#VIBRATE) | Allows access to the vibrator  |
| [WAKE\_LOCK](http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf#WAKE_LOCK) | Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming  |
| [WRITE\_APN\_SETTINGS](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_APN_SETTINGS) | Allows applications to write the apn settings  |
| [WRITE\_CALENDAR](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_CALENDAR) | Allows an application to write (but not read) the user's calendar data. |
| [WRITE\_CONTACTS](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_CONTACTS) | Allows an application to write (but not read) the user's contacts data. |
| [WRITE\_EXTERNAL\_STORAGE](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_EXTERNAL_STORAGE) | Allows an application to write to external storage  |
| [WRITE\_GSERVICES](http://developer.android.com/reference/android/app/ActivityManager.html#WRITE_GSERVICES) | Allows an application to modify the Google service map. |
| [WRITE\_HISTORY\_BOOKMARKS](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_HISTORY_BOOKMARKS) | Allows an application to write (but not read) the user's browsing history and bookmarks. |
| [WRITE\_SECURE\_SETTINGS](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_SECURE_SETTINGS) | Allows an application to read or write the secure system settings. |
| [WRITE\_SETTINGS](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_SETTINGS) | Allows an application to read or write the system settings. |
| [WRITE\_SMS](http://developer.android.com/reference/android/Manifest.permission.html#WRITE_SMS) | Allows an application to write SMS messages. |
| [WRITE\_SYNC\_SETTINGS](http://download.oracle.com/javase/6/docs/api/java/security/KeyStore.htm#WRITE_SYNC_SETTINGS) | Allows applications to write the sync settings  |

###### Compliance Checklist

|  |  |  |
| --- | --- | --- |
| Section | Description | DIACAP or STIG Reference |
| 2.1 | When requesting any Android permission, the developer must provide documentation as to the need for the permission. |  |
| 2.4 | Passwords should be masked so that they are not visibly displayed. |  |
| 2.4 | Passwords must be at least # characters long. |  |
| 2.4 | Insert additional password requirements. |  |
| 2.4 | Passwords must be encrypted in transit when transmitted. |  |
| 2.4 | Mobile devices require the use of a standard Notice and Consent banner, the contents of which are detailed in this section. The banner shall be implemented as a click-through at logon to the device or application. |  |
| 3.1 | Sensitive and personally identifiable information (PII) shall be protected by means of FIPS 140-2 compliant cryptographic algorithms and implementations. |  |
| 3.2 | Sensitive data should not be stored in the /sdcard partition. |  |
| 3.4 | Sensitive data should not be stored in files with world-read or world-write permissions. |  |
| 4.1 | Text fields should be strongly typed, meaning that the allowed characters, lengths, and ranges are known and enforced through input validation. |  |
| 4.1.1 | User input should not be directly passed to SQL queries. Instead, prepared or parameterized statements should be used for SQL queries. |  |
| 4.1.2 | The Java Class.forName, Class.newInstance, and Runtime.exec method calls should be avoided, with native Java calls used instead of making system calls when possible. If these methods need to be used, the developer must justify their use in documentation. |  |
| 4.2 | All Android application packages must be signed by a private key associated with a certificate held by the application developer or a trusted authority. |  |
| 4.3 | Application developers should avoid use of the Android NDK or Java JNI unless their use is necessary. Developers must justify in writing any use of the NDK or JNI. |  |
| 5 | Mobile applications should use data-in-transit encryption for all transmitted information unless a specific reason exists not to, which must be justified in writing. |  |
| 5.1.1 | SSL version 2.0 must not be used. |  |
| 5.1.1 | TLS versions 1.0, 1.1, 1.2, or above should be used. |  |
| 5.1.1 | In RSA PKI certificate environments, the TLS recommended cipher suites are:TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHATLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHATLS\_RSA\_WITH\_AES\_256\_CBC\_SHATLS\_RSA\_WITH\_AES\_128\_CBC\_SHA |  |
| 5.1.1 | In elliptic curve PKI certificate environments, the TLS recommended cipher suites are:TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHATLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA |  |
| 5.1.2 | Android applications should be configured to only trust the appropriate PKI root certification authorities that are needed for communication by that application, rather than using the default Android root certificate list. |  |
| 5.1.3 | FIPS validated cryptographic implementations should be used. |  |
| 5.1.4 | TLS clients must always authenticate the server's certificate. |  |
| 5.1.4 | TLS client certificate authentication should be used when possible as it is a strong authentication method. |  |
| 5.1.4 | When username and password authentication is used, storing passwords on the Android device is prohibited. Instead, the client should obtain an authentication token from the server and store that instead if needed. |  |
| 5.1.5 | When IPsec VPNs are used, the guidance in NIST Special Publication 800-77 and 800-57 Part 3 should be followed. A FIPS validated IPsec client should be used. |  |
| 5.2 | Sensitive information such as phone number,device serial number, and device ID should not be used to identify connections. |  |
| 6.1.1.1 | Applications should use explicit Intents whenever it is possible to address a specific destination component. |  |
| 6.1.1.1 | Applications must use explicit Intents if the Intent is for an internal component (intra-application communication). |  |
| 6.1.1.1 | When sending broadcast Intents, applications should define a custom permission (or use existing appropriate permission) and only send the Intent to receivers that hold the permission in order to control who can receive the Intent. If all receiving applications are written by the same developer and signed by the same key, the permission should be declared as a signature-level permission. |  |
| 6.1.1.1 | Especially when implicit Intents are used, applications should avoid placing sensitive information inside the Intent. Applications should also perform input validation on any received response. |  |
| 6.1.1.2 | Before sending an explicit Intent, an application should check that the receiving application holds an appropriate permission to receive the Intent. In especially sensitive environments, an application should check the receiving application's package signature to ensure that it matches an expected value. |  |
| 6.1.2 | Internal application components (that have no need to be invoked by other applications) should explicitly be declared as not exported in the manifest |  |
| 6.1.2 | Applications must perform null checks on all received Intents. |  |
| 6.1.2 | Applications must ensure that received broadcast messages match an expected action before continuing processing them. |  |
| 6.1.2 | Applications should set permissions on their external components to control which other applications can invoke them. In especially sensitive environments, the application should obtain the sending application's package signature to ensure that it matches an expected value. |  |
| 6.1.3 | Sticky Broadcasts should not be used to send Intents with sensitive information. |  |
| 6.2 | Content Providers must only be exported when other applications have a need to use them. |  |
| 6.2 | If a Content Provider is exported, the application developer must define read and write permissions to prevent the Content Provider from being misused by malicious applications. |  |
|  |  |  |

###### Acronyms

AES Advanced Encryption Standard

APK (Android) Application Package

ATO Authority to Operate

C&A Certification and Accreditation

CA Certification Authority

CAC Common Access Card

CBC Cypher-Block Chaining

COMSEC Communications Security

CVE Common Vulnerabilities and Exposures

DAA Designated Approving Authority

DARPA Defense Advanced Research Projects Agency

DBA Database Administrator

DHE Diffie-Hellman (key exchange)

DIACAP DoD Information Assurance Certification and Accreditation Process

DISA Defense Information Systems Agency

DoD Department of Defense

DoDI Department of Defense Instruction

ECDHE Elliptic-curve Diffie-Hellman

ECDSA Elliptic-curve Digital Signature Algorithm

FIPS Federal Information Protection Standard

GIG Global Information Grid

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IA Information Assurance

IAM Information Assurance Manager

IAT Information Assurance Technical

IETF Internet Engineering Task Force

IP Internet Protocol

IPT Integrated Project Team

IT Information Technology

JNI Java Native Interface

NDK (Android) Native Development Kit

NIPRNET Non-secure Internet Protocol Router Network

NIST National Institute of Standards and Technology

NSA National Security Agency

OS Operating System

PED Portable Electronic Device

PII Personally Identifiable Information

PIT Platform Information Technology

PKI Public Key Infrastructure

PIV Personal Identification Verification

RDBMS Relational Database Management System

RFC (IETF) Request for Comment

RSA Rivest, Shamir and Adleman (algorithm for PKI)

SD Secure Digital (Card)

SEE SQLite Encryption Extension

SHA Secure Hash Algorithm

SQL Structured Query Language

STIG Security Technical Implementation Guide

SDK Software Development Kit

SMS Short Messaging System

SSL Secure Sockets Layer

TLS Transport Layer Security

US United States

VPN Virtual Private Network

1. [http://iase.disa.mil/stigs/stig/index.html](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-1)
2. Jesse Burns, iSEC Partners, Developing Secure Mobile Applications for Android, October 2008, Version 1.0 [↑](#footnote-ref-2)
3. http://developer.android.com/reference/android/Manifest.permission.html [↑](#footnote-ref-3)
4. Application Security and Development STIG, V3R2, 3.8 Authentication, Figure 3-2, DISA Field Security Operations 29 October 2010 [↑](#footnote-ref-4)
5. Application Security and Development STIG, V3R2, 3.8.1.2 User Authentication, DISA Field Security Operations 29 October 2010 [↑](#footnote-ref-5)
6. Application Security and Development STIG, V3R2, 3.8.2.1 Standalone Application Authentication 29 October 2010 [↑](#footnote-ref-6)
7. Application Security and Development STIG, V3R2, 3.8.4 Password Authentication, DISA Field Security Operations 29 October 2010 [↑](#footnote-ref-7)
8. Draft Android 2.2 (Dell) Technology Overview, V1R0.1, E.2 Provisioning Procedures, DISA Field Security Operations 01 June 2011 [↑](#footnote-ref-8)
9. Application Security and Development STIG, V3R2, 3.8.8 Logon Banner, DISA Field Security Operations 29 October 2010 [↑](#footnote-ref-9)
10. http://www.hwaci.com/sw/sqlite/see.html [↑](#footnote-ref-10)
11. http://sqlite-crypt.com [↑](#footnote-ref-11)
12. http://sqlcipher.net [↑](#footnote-ref-12)
13. http://code.google.com/p/android/issues/detail?id=191 [↑](#footnote-ref-13)
14. [http://www.oracle.com/technetwork/java/seccodeguide-139067.html](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-14)
15. [http://www.isecpartners.com/files/iSEC\_Securing\_Android\_Apps.pdf](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-15)
16. Application Security and Development STIG, V3R2, 3.8.1.3 Authentication, Figure 3-2, DISA Field Security Operations 29 October 2010 [↑](#footnote-ref-16)
17. [http://developer.android.com/sdk/ndk/overview.html](http://developer.android.com/reference/android/Manifest.permission.html), retrieved on 9 June 2011. [↑](#footnote-ref-17)
18. Adapted from [http://code.google.com/p/android/issues/detail?id=16041](http://developer.android.com/reference/android/app/ActivityManager.html?id=16041) and [http://download.oracle.com/javase/6/docs/api/java/security/KeyStore.htm](http://developer.android.com/reference/android/Manifest.permission.html)l [↑](#footnote-ref-18)
19. Enck, et al. A Study of Android Application Security. http://www.cse.psu.edu/~swarat/pubs/enck-sec11.pdf [↑](#footnote-ref-19)
20. Chin, et al. Analyzing Inter-Application Communication in Android. MobiSys 2011. [http://www.cs.berkeley.edu/~afelt/intentsecurity-mobisys.pdf](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-20)
21. [http://www.enck.org/pubs/enck-sec11.pdf](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-21)
22. http://www.isecpartners.com/files/iSEC\_Securing\_Android\_Apps.pdf [↑](#footnote-ref-22)
23. iSEC Partners, Developing Secure Mobile Applications for Android. [http://www.isecpartners.com/files/iSEC\_Securing\_Android\_Apps.pdf](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-23)
24. From Chin, et al [↑](#footnote-ref-24)
25. From Chin, et al [↑](#footnote-ref-25)
26. From Chin, et al [↑](#footnote-ref-26)
27. Jesse Burns, iSEC Partners, Developing Secure Mobile Applications for Android. [http://www.isecpartners.com/files/iSEC\_Securing\_Android\_Apps.pdf](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-27)
28. Jesse Burns, iSEC Partners, Developing Secure Mobile Applications for Android. [http://www.isecpartners.com/files/iSEC\_Securing\_Android\_Apps.pdf](http://developer.android.com/reference/android/Manifest.permission.html) [↑](#footnote-ref-28)