Android security white paper

Last updated: April 2016

Contents

About this document

This white paper provides an overview of various security features that are in place—at the OS level and at the Google services layer. It also introduces the new device management capabilities developed for work, which give enterprises the ability to manage a workspace on their users' devices, prevent work data leakage, secure the communication back to the enterprise, and manage the applications installed in their workspace, preventing any unapproved apps from being installed for work.

Introduction

The Android operating system leverages traditional OS security controls to protect user data and system resources, protects device integrity against malware, and provides application isolation. Additionally, Google provides a number of services layered on top of the OS that, when combined with Android OS security, help to continuously protect the Android user.

Android OS

Android is an open source OS that's built on the Linux® kernel and provides an environment for multiple applications to run simultaneously. These applications are signed and isolated into application sandboxes associated with their application signature. The application sandbox defines the privileges available to the application. Applications are generally built using Android Runtime and interact with the OS through a framework that describes system services, platform Application Programming Interfaces (APIs), and message formats. Other high-level languages (for example, JavaScript®) and lower-level languages (for example, ARM® assembly) are allowed and operate within the same application sandbox. System services are implemented as applications and are constrained by an application sandbox. Above the kernel, there's no concept of a superuser or root that has unconstrained access to the system.

Figure 1 summarizes the security components and considerations of the various levels of the Android OS.

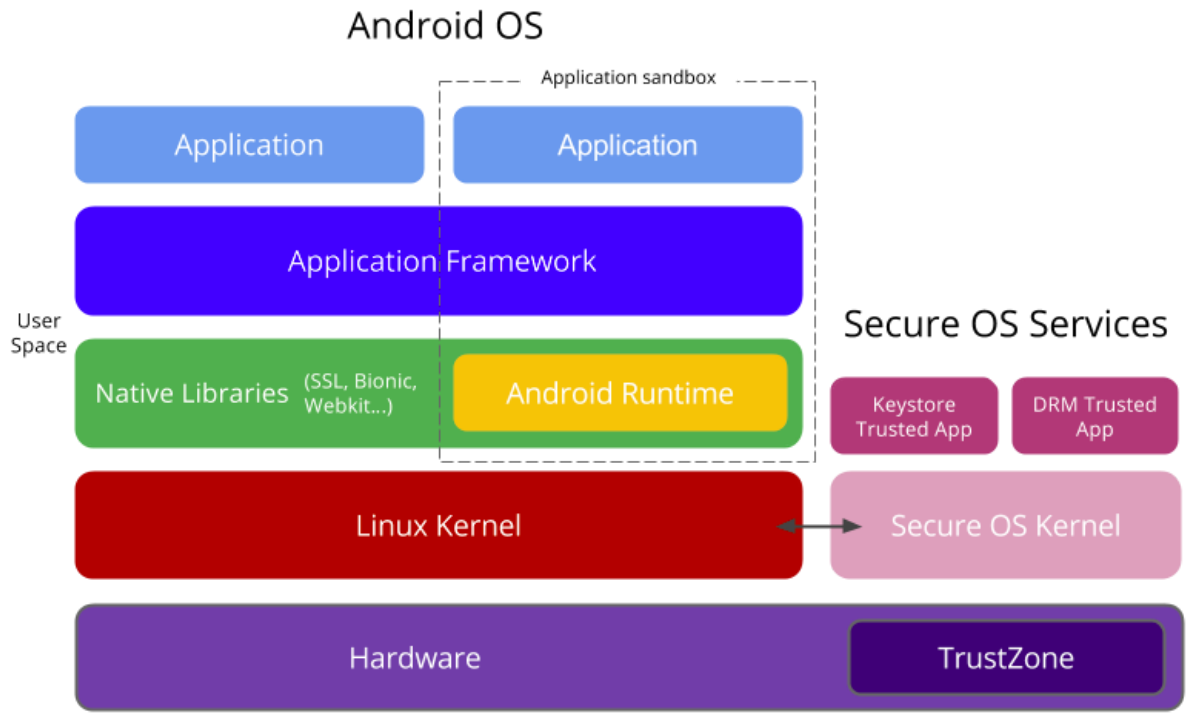

Figure 1. Android security architecture on ARM with TrustZone support

Android Secure OS services

Android is a multipurpose operating system. Many Android devices provide a secondary, isolated environment to run privileged or security-sensitive operations that don't need the functionality of a multipurpose OS. This environment is sometimes referred to as a *Secure OS*. These capabilities can be implemented on a separate processor (such as a standalone Secure Element or Trusted Platform Module [TPM]), or can be isolated beneath the kernel on a shared processor (such as ARM TrustZone® technology).

The Secure OS can be used by the original equipment manufacturer (OEM) to provide device-specific services and applications. Most Android devices implement Widevine DRM-protected video playback services within the Secure OS. Starting with Android 4.3, cryptographic services based in the SecureOS have also been exposed to Android applications via the KeyChain API. This API provides the ability for applications to create keys that cannot be exported, even in the event of an Android compromise.

Cryptography and data protection

Cryptography is used throughout Android to provide confidentiality and integrity. Google supports most of the industry-standard algorithms. The following lists major uses of cryptography on Android:

- Device encryption
- Application signing
- Network connectivity and encryption, including SSL, Wi-Fi, and VPN

Device encryption

Encryption is the process of encoding user data on an Android device using an encrypted key. Once a device is encrypted, all user-created data is automatically encrypted before committing it to disk and all reads automatically decrypt data before returning it to the calling process.

Android disk encryption is based on **dm-crypt**, which is a kernel feature that works at the block device layer. The encryption algorithm is 128 Advanced Encryption Standard (AES) with cipher-block chaining (CBC) and ESSIV:SHA256. The master key is encrypted with 128-bit AES via calls to the Android OpenSSL library. OEMs can use 128-bit or higher to encrypt the master key.

Android 5.0 introduces the following new encryption features:

- Fast encryption, which only encrypts used blocks on the data partition to avoid first boot taking a long time.
- Added the **forceencrypt** flag to encrypt on first boot.
- Added support for patterns and encryption without a password.
- Added hardware-backed storage of the encryption key.

In the Android 5.0 release, there are four kinds of encryption states:

- Default
- PIN
- Password
- Pattern

If default encryption is enabled on a device, then upon first boot, the device generates a 128-bit key, which is then encrypted with a default password, and the encrypted key is stored in the crypto metadata. Hardware backing is implemented by using the Trusted Execution Environment's (TEE's) signing capability. The generated 128-bit key is valid until the next factory reset (i.e. until the `/data` partition is erased). Upon factory reset, a new 128-bit key is generated.

When the user sets the PIN or password on the device, only the 128-bit key is re-encrypted and stored (i.e. user PIN/Password/Pattern changes don't cause re-encryption of user data).
The Android 5.0 Compatibility Definition Document (CDD) requires that if a device implementation has a lock screen, the device must support full-disk encryption of the application private data; that is, the `/data` and the SD card partition, if it's a permanent, non-removable part of the device.

Notes:

1. The encryption key must not be written to storage at any time without being encrypted. Other than when in active use, the encryption key must be AES-encrypted with the lock screen passcode stretched, using a slow stretching algorithm. If the user hasn't specified a lock screen passcode or has disabled passcode use for encryption, the system uses a default passcode to wrap the encryption key. If the device provides a hardware-backed keystore, the password stretching algorithm must be cryptographically bound to that keystore.
2. Devices encrypted at first boot cannot be returned to an unencrypted state after factory reset.

KeyChain and KeyStore

Android provides a set of cryptographic APIs for use by applications. These APIs include implementations of standard and commonly used cryptographic primitives, such as AES, Rivest-Shamir-Adleman (RSA), Digital Signature Algorithm (DSA), and Secure Hash Algorithm (SHA). Additionally, APIs are provided for higher-level protocols, such as Secure Socket Layer (SSL) and HTTPS.

Android 4.0 introduced the KeyChain class to allow applications to use the system credential storage for private keys and certificate chains. The KeyChain API is used for Wi-Fi and Virtual Private Network (VPN) certificates.

The Android KeyStore class lets you store private keys in a container to make it more difficult to extract from the device. It was introduced in Android 4.3 and focuses on applications storing credentials used for authentication, encryption, or signing purposes.

Applications can call isBoundKeyAlgorithm in KeyChain before importing or generating private keys of a given algorithm, to determine if hardware-backed keystore is supported to bind keys to the device in a way that makes them non-exportable.

Application security

Applications are an integral part of any mobile platform and users increasingly download applications to their devices. Android provides multiple layers of application protection, enabling users to download their favorite applications to their devices with the peace of mind that they're getting a high level of protection from malware, security exploits, and attacks. The following subsections define the main Android application security features.

Application sandbox and permissions

Android applications run in what is referred to as an *application sandbox*. Just like the walls of a sandbox keep the sand from getting out, each application is housed within a virtual *sandbox* to keep it from accessing anything outside itself. By default, some applications need to use functionality on the device that isn't in the sandbox; for example, accessing contact information. Before installing an application, determine whether or not the user can grant *permission* to the app to access certain capabilities on the device (for example, *Make phone calls*). A phone dialer application should naturally be able to make phone calls. On the flip side, if the application is supposed to be a puzzle

game, that same request might look a bit more suspicious. By providing these details upfront, users can make an educated decision about trusting an app or not.

The Android platform takes advantage of the Linux user-based protection as a means of identifying and isolating application resources. The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process. This approach is different from other operating systems (including the traditional Linux configuration), where multiple applications run with the same user permissions.

This sets up a kernel-level application sandbox. The kernel enforces security between applications and the system at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications. By default, applications can't interact with each other and applications have limited access to the OS. For example, if application A tries to do something malicious like read application B's data or dial the phone without permission (which is a separate application), then the OS protects against this because application A doesn't have the appropriate user privileges. The sandbox is simple, auditable, and based on decades-old, UNIX-style user separation of processes and file permissions.

Because the application sandbox is in the kernel, this security model extends to native code and to OS applications. All of the software above the kernel in Figure 1 (including OS libraries, application framework, application runtime, and all applications) run within the application sandbox.
On some platforms, developers are constrained to a specific development framework, set of APIs, or language to enforce security. On Android, there are no restrictions on how an application can be written that are required to enforce security; native code is just as secure as interpreted code.
In some operating systems, memory corruption errors generally lead to completely compromising the security of the device. This is not the case in Android due to all applications and their resources being sandboxed at the OS level. A memory corruption error only allows arbitrary code execution in the context of that particular application, with the permissions established by the OS.

Security Enhanced Linux

As part of the Android security model, the Android sandbox also uses Security Enhanced Linux (SELinux) to enforce Mandatory Access Control (MAC) over all processes, even processes running with root and superuser privileges. SELinux provides a centralized analyzable policy and strongly separates processes from one another.

Android includes SELinux in enforcing mode (for example, security policy is enforced and logged) and a corresponding security policy that works by default across Android Open Source Project (AOSP). In enforcing mode, illegitimate actions that violate policy are prevented and all violations (denials) are logged by the kernel to dmesg and logcat.

The Android 5.0 CDD mandates that devices must implement a SELinux policy that allows the SELinux mode to be set on a per-domain basis, and all domains configured in enforcing mode. No permissive mode domains are allowed. The Compatibility Test Suite (CTS) for SELinux ensures security policy compatibility and enforces security best practices.

Application signing

Android requires that all apps be digitally signed with a certificate before they can be installed. The certificate doesn't need to be signed by a certificate authority. Android uses this certificate to identify the author of the application. Android applications often use self-signed certificates and the application developer holds the certificate's private key. When the system installs an update to an application, it compares the certificate in the new version with those in the existing version, and allows the update if the certificate matches.

Android allows applications signed by the same certificate to run in the same process, if the applications so request, so that the system treats them as a single application. Android provides signature-based permissions enforcement, so that an application can expose functionality to another app that's signed with a specified certificate. By signing multiple apps with the same certificate, and using signature-based permissions, an app can share code and data in a secure manner.

The key must have a validity period that exceeds the expected lifespan of the app. (A validity period of 25 years or more is recommended.) When a key's validity period expires, users can no longer seamlessly upgrade to new versions of the application.

Note: Applications published on Google Play must be signed with keys that have a validity period ending after October 22, 2033. Google Play enforces this requirement to ensure that users can seamlessly upgrade apps when new versions are available.

Google Play app review

Google Play is Android's app distribution platform that protects users from potentially harmful apps. Google Play has policies in place to protect users from attackers trying to distribute potentially harmful apps. Within Google Play, developers are validated in two stages. Developers are first reviewed when they create their Google Play developer account based on their profile and credit cards. Developers are then reviewed further with additional signals upon app submission. Google regularly scans Play applications for malware and other vulnerabilities. Google also suspends developer accounts that violate developer program policies.

Google Play also has rating and reviews that provide information about an application before installing it. If an app tries to mislead users, it's likely to have a low star rating and poor comments.

An example of Google's developer security advocacy, was for apps running vulnerable versions of the Apache Cordova™ platform. Google notified:

- Developers via the Google Play Developer Console and email
- Developers of apps containing private keys or keystore files

Verify apps

Android devices that have Google Play installed have the option of using Google's Verify Apps feature, which scans apps when you install them and periodically scans for potentially harmful apps. App verification is turned on, by default, but no data is sent to Google, unless the user agrees to allow this when prompted in the dialog box, prior to installing the first app from a source other than Google Play.

Verify Apps is available on Android 2.3+ with Google Play. On devices running Android 4.2 or higher, users can enable or disable Verify Apps from **Google Settings > Security > Verify Apps**.
Verify Apps now continually checks devices to ensure that all apps behave in a safer manner, even after installation. This enhancement takes the protection even further, using Android's powerful app scanning system developed by the Android Security and Safe Browsing teams.

Network security

In addition to data-at-rest security protecting information stored on the device, Android provides network security for data-in-transit to protect data sent to and from Android devices. Android provides secure communications over the Internet for web browsing, email, instant messaging, and other Internet applications, by supporting Transport Layer Security (TLS), including TLS v1.0, TLS v1.1, TLS v1.2, and SSL v3.

Wi-Fi

Android supports the WPA2-Enterprise (802.11i) protocol, which is specifically designed for enterprise networks and can be integrated into a broad range of Remote Authentication Dial-In User Service (RADIUS) authentication servers. The WPA2-Enterprise protocol support uses AES-128 encryption in Android 5.0, thus providing corporations and their employees a high level of protection when sending and receiving data over Wi-Fi.

Android supports 802.1x Extensible Authentication Protocols (EAPs), including EAP-TLS, EAP-TTLS, PEAPv0, PEAPv1, and EAP-SIM, introduced in Android 5.0.

VPN

Android supports network security using VPN:

- **Always-on VPN**—The VPN can be configured so that applications don't have access to the network until a VPN connection is established, which prevents applications from sending data across other networks.
- **Per User VPN**—On multiuser devices, VPNs are applied per Android user, so all network traffic is routed through a VPN without affecting other users on the device.
- **Per Profile VPN**—VPNs are applied per Work Profile, which allows an IT administrator to ensure that only their enterprise network traffic goes through the enterprise-Work Profile VPN—not the user's personal network traffic.
- **Per Application VPN**—Android 5.0 provides support to facilitate VPN connections on allowed applications or prevents VPN connections on disallowed applications.

Third-party applications

Google is committed to increasing the use of TLS/SSL in all applications and services. As applications become more complex and connect to more devices, it's easier for applications to introduce networking mistakes by not using TLS/SSL correctly.

The Android Security team has built a tool called *nogotofail*, which provides an easy way to confirm that devices or applications are safe against known TLS/SSL vulnerabilities and misconfigurations.

The nogotofail tool works for Android and other operating systems. There's an easy-to-use client to configure the settings and get notifications on Android. The nogotofail tool is released as an open source project so application developers can test their applications, contribute new features to the project, and help improve the network security on Android.

Device and profile management

Android 5.0 introduces the concept of a Device Owner and Profile Owner to support the corporate owned and bring your own device (BYOD) enterprise uses cases, respectively. The concept of a Managed Profile is based on the Android multiuser concept, first introduced in Android 4.2 (API 17).

Android users

An Android user is intended to be used by a different physical person and has their own application data, some unique settings, and UI to explicitly switch between them. A user can run in the background when another user is active. A user's data is always isolated from other users. Android supports Primary and Secondary users as defined below:

- A **Primary user** is the first user added to a device. It can't be removed, except by factory reset. This user also has special privileges and settings only set by that user. The Primary user is always running even when other users are in the foreground.
- A **Secondary user** is any user added to the device other than the Primary user. A secondary user can be removed by their own doing and by the primary user, but can't impact other users on a device. Secondary users can run in the background and continue to have network connectivity when they do. However, there are some restrictions; for example, not being able to display UI or have Bluetooth™ services active while in the background. Background secondary users are halted by the system process if the device requires additional memory for operations in the foreground user.

Managed Profile

A Device Policy Client (DPC) is an application used to manage the corporate space on the device. The DPC has access to the device management APIs available in the DevicePolicyManager class and receives callbacks from the system via the DeviceAdminReceiver class.

A **Work Profile** is a managed profile created when the DPC initiates a managed provisioning flow. In this instance, a Work Profile functions like a regular user, but is associated with the primary user in such a way that notifications and the recent task list are shared. Applications, notifications and widgets from the Managed Profile are always badged. Because the Work Profile is a separate Android user, there's a strong separation between the corporate and personal profile, and all data within the Work Profile is managed separately by the enterprise.

A **Profile Owner** is a special case of a device administrator, who can only manage the corporate space on a user's personal device to support the BYOD use case. Profile owners are scoped to the Work Profile and can only be defined as part of the managed provisioning process. The user experience is enhanced to allow the user to easily access both personal and Work Profiles at once. The Profile Owner can't be deactivated by the user; however, the user is always able to view and validate the settings being enforced within the Work Profile. The user can choose to remove the Work Profile and the Profile Owner altogether whenever they desire.

A **Device Owner** is like a Profile Owner, but scoped to the whole device. The Device Owner is the device administrator in the corporate-owned device use case.

Cross profile intents

In the BYOD case, data in the Work Profile is segregated from the user's personal data. However, there are instances where allowing intents from one profile to be resolved in the other can be useful and enhance the enterprise user's productivity. In the Work Profile, IT administrators control sharing between managed and personal profiles. Two new methods have been added in Android 5.0 to DevicePolicyManager class for cross profile intents: addCrossProfileIntentFilter and clearCrossProfileIntentFilters.

By default, the following intents are automatically configured by the system during the Work Profile creation to be forwarded to the Primary Profile:

- Telephony intents
- Mobile network settings
- Home intent—The launcher doesn't run in the Work Profile.
- Get content—The user has the option to resolve in either the Primary or Work Profile.
- Open document—The user has the option to resolve in either the Primary or Work Profile.
- Picture—The user has the option to resolve in either the Primary or Work Profile if an app that can handle camera exists in the Work Profile.
- Set clock—The user has the option to resolve in either the Primary or Work Profile.
- Speech recognition—The user has the option to resolve in either the Primary or Work Profile.

Additionally, the SEND intent, used when sharing content, is configured to offer the user the option to forward the content into the Work Profile.

Note: The SEND intent is **not** automatically configured to offer the user the option to forward their content *from* the Work Profile into the primary because some IT administrators consider this a security risk. Instead, the DPC application has the option of adding this functionality, if allowed by a company's IT policy.

Device and profile policies

Android 5.0 adds a number of security policies and configurations for both device and profile management. IT administrators can set these policies (indirectly) via a mobile device management (MDM) solution to secure work data on their employees' devices. The following table lists these policies, indicating whether they apply to devices for corporate-owned device cases or profile for BYOD cases.

Policy	Device	Profile
addCrossProfileIntentFilter		✓
addCrossProfileWidgetProvider		✓
addPersistentPreferredActivity	✓	✓
addUserRestriction	✓	✓
clearCrossProfileIntentFilters		✓
clearDeviceOwnerApp	✓	
clearPackagePersistentPreferredActivities	✓	✓
clearUserRestriction	✓	✓
createAndInitializeUser	✓	
enableSystemApp	✓	✓
installCaCert	✓	✓
installKeyPair	✓	✓
lockNow	✓	✓
removeActiveAdmin	✓	✓
removeCrossProfileWidgetProvider		✓
removeUser	✓	

resetPassword	✓	✓
setAccountManagementDisable	✓	✓
setApplicationHidden	✓	✓
setApplicationRestrictions	✓	✓
setAutoTimeRequired	✓	
setCameraDisabled	✓	✓
setCrossProfileIdDisabled		✓
setGlobalSetting	✓	
setKeyguardDisabledFeatures	✓	✓
setLockTaskPackages	✓	
setMasterVolumeMuted	✓	✓
setMaximumFailedPasswordsForWipe	✓	✓
setMaximumTimeToLock	✓	✓
setPasswordExpirationTimeout	✓	✓
setPasswordHistoryLength	✓	✓
setPasswordMinimumLength	✓	✓
setPasswordMinimumLetters	✓	✓
setPasswordMinimumLowerCase	✓	✓
setPasswordMinimumNonLetter	✓	✓
setPasswordMinimumNumeric	✓	✓
setPasswordMinimumSymbols	✓	✓
setPasswordMinimumUpperCase	✓	✓
setPasswordQuality	✓	✓
setPermittedAccessibilityServices	✓	✓

setPermittedInputMethods	✓	✓
setProfileEnabled		✓
setProfileName		✓
setRecommendedGlobalProxy	✓	✓
setRestrictionsProvider	✓	✓
setScreenCaptureDisabled	✓	✓
setSecureSetting	✓	✓
setStorageEncryption	✓	✓
setUninstallBlocked	✓	✓
switchUser	✓	
uninstallAllUserCaCerts		✓
uninstallCaCert	✓	✓
wipeData	✓	✓

Application management

Android's enterprise features create a secure framework for companies to put any application in Google Play to work for them in a simple, standard way. Through managed Google Play, an enterprise version of Google Play, IT administrators can easily find, deploy, and manage work applications while ensuring malware and other threats are neutralized.

Google Play

Managed Google Play provides APIs for use by Enterprise Mobility Management (EMM) vendors to allow them to manage applications on devices in an Android enterprise domain. The APIs provide functionality for use (indirectly) by administrators of the enterprises managed by the EMM as follows:

- An IT administrator can remotely install or remove apps on managed Android enterprise devices via the EMM's app. This action is limited to devices or profiles that are managed by the EMM's app, which ensures that the user has consented to the EMM's access.
- An IT administrator can define which users should be able to see which apps. A user running the Play Store app within the Work Profile only sees the apps visible to them.
- Enterprise administrators can see which users have apps installed or provisioned, and the number of licenses purchased and provisioned.

Installation of applications within the Work Profile is possible via managed Google Play in the Work Profile, either by direct user request in the managed Play Store app (pull), or as a result of a call to the EMM API (push). When the user opens the Play Store app in the Work Profile, it only displays the apps which the IT administrator has specified the user can access. The user can install these applications, but not others.

Secure app serving

Transport of all Android application packages (APKs) and app metadata between Google Play and Android devices is encrypted using SSL. App access is authenticated and authorized using the Google Account created as part of user registration in the Android enterprise domain.

Private apps

With managed Google Play, apps can be published by an enterprise customer and targeted privately (i.e. they're only visible and installable by users within that enterprise's Android enterprise domain). Private apps are logically separated in Google's cloud infrastructure from Google Play for consumers. There are two modes of delivery for private apps:

- **Google hosted**—By default, Google hosts the APK in its secure data centers.
- **externally-hosted**—Enterprise customers host APKs on their own servers—accessible only on their intranet or via VPN. Details of the requesting user and their authorization is provided via a JSON Web Token (JWT) with an expiry time. The JWT is signed by Google using

the key pair associated with the specific app in Play, and should be verified before trusting the authorization contained in the JWT.

In both cases, Google Play stores the app metadata—title, description, graphics, and screenshots. Apps must comply with all Google Play policies in all cases.

Unknown sources

By default, the Unknown sources setting under **Settings > Security > Unknown sources** is off. The Device Owner or Profile Owner can disable user control of Unknown sources in the Managed Device or Work Profile by setting the DISALLOW_INSTALL_UNKNOWN_SOURCES user restriction to **True** using addUserRestriction. The default value for DISALLOW_INSTALL_UNKNOWN_SOURCES user restriction in both Device Owner and Profile Owner is false. When DISALLOW_INSTALL_UNKNOWN_SOURCES is set to true by the Device Owner or Profile Owner, the user cannot modify the Unknown sources security setting on the device or Work Profile; however, in the case of Work Profile, the user can still modify Unknown sources setting in their personal space.

Additionally, the side loading of applications using Android Debug Bridge (adb) can be disabled via the DISALLOW_DEBUGGING_FEATURES user restriction in a Managed Device by Device Owner, or Work Profile by Profile Owner. The default value of DISALLOW_DEBUGGING_FEATURES for both Device Owner and Profile Owner is false.

Setting DISALLOW_INSTALL_UNKNOWN_SOURCES and DISALLOW_DEBUGGING_FEATURES user restrictions to **True** by EMMs, provides an extra measure of assurance to IT administrators that only company-approved apps will be deployed using Google Play to users in a corporate-managed device or profile.

Managed App configuration

Android in the enterprise provides the ability to set policies on a per-application basis, where the app developer has made this available. For example, an app could allow an IT administrator to remotely control the availability of features, configure settings, or set in-app credentials. The **setApplicationRestrictions** method allows EMMs to configure these restrictions via the DevicePolicyManager class.

Google Chrome is an example of an enterprise-managed app that implements policies and configurations that can be fully managed according to enterprise policies and restrictions.

Security best practices

Google designed Android and Google Play to provide everyone with a safer experience. With that goal in mind, the Android Security team works hard to minimize the security risks on Android devices. Google's multilayered approach starts with prevention and continues with malware detection and rapid response should any issues arise.

More specifically, Google:

- Strives to **prevent** security issues from occurring through design reviews, penetration testing and code audits
- Performs security reviews prior to releasing new versions of Android and Google Play
- Publishes the source code for Android, thus allowing the broader community to uncover flaws and contribute to making Android the most secure mobile platform
- Works hard to **minimize** the impact of security issues with features like the application sandbox
- **Detects** vulnerabilities and security issues by regularly scanning Google Play applications for malware, and removing them from devices if there's a potential for serious harm to the user devices or data
- Has a rapid response program in place to handle vulnerabilities found in Android by working with hardware and carrier partners to quickly resolve security issues and push security patches

The Android team works very closely with the wider security research community to share ideas, apply best practices, and implement improvements. Android is part of the Google Patch Reward Program, which pays developers when they contribute security patches to popular open source projects, many of which form the foundation for AOSP. Google is also a member of the Forum of Incident Response and Security Teams (FIRST).

Conclusion

For a long time, being secure has been synonymous with being closed. But the mobile ecosystem is now transitioning from closed, isolated platforms towards open platforms that foster innovation and allow interoperability with confidence. Android gains security from being more open. Android's security is built to protect its users in a complex ecosystem that includes system-on-a-chip vendors (SoCs), OEMs, service providers, independent software vendors (ISVs), and enterprises, just to name a few.

Google's commitment to security for all Android users includes a combination of built-in security features in the platform (such as application sandboxing) and Google services-based protections (such as Google Play and Verify apps). Behind Google Play's attempt to protect against potentially harmful applications is a vast, systemic knowledge of Android applications accumulated over many years, beginning with the onset of Android. Google Play uses a combination of static, dynamic, and relationship analysis, combined with thousands of unique signals to analyze each application. Every application on Google Play is reviewed through a combination of technology, human review, and user community flags.

Finally, Android 5.0 enhances Android device management capabilities by introducing Work Profiles. In the context of Android for business, enterprises rely on managed Google Play for deploying applications. Unknown sources and third-party marketplaces can be disallowed by EMMs, thus protecting employees' devices from any potential malicious applications to be installed in the Work Profile.