Android Enterprise Security White Paper

Last updated: September 2018

Contents

About this document

Mobile security threats are a major concern for enterprises, especially as employees often use their devices every day to access corporate systems and data. Enterprises require, and demand, robust security protections - protections which Android offers comprehensively.

This security white paper outlines the approach Android takes to mobile security for business customers, and details the strengths of the Android platform, the range of management APIs available to enforce control, and the role of Google Play Protect in detecting threats.

Android offers a multi-layer security strategy with unique ways to keep data and devices safe. Beyond hardware and OS-protections, Android offers multi-profile support and device-management options that enable the separation of work and personal data, keeping company data secure.

To help ensure threats are detected before they even make it onto the device, Google Play Protect provides built-in continuous scanning for apps and automatic removal of harmful apps.

This white paper will also explain how the open source Android platform enables best-in-class enterprise security by leveraging the collective intelligence of the Android ecosystem.

Android system architecture

Android OS

Android is an open source OS that's built on the Linux® kernel and provides an environment for multiple apps to run simultaneously. These apps are signed and isolated into application sandboxes associated with their application signature. The application sandbox defines the privileges available to the application. Apps 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 and lower-level languages, such as C/C++, are allowed and operate within the same application sandbox.

Trusted Execution Environment

Android devices that support a lock screen and ship with Android 7.0 Nougat and higher have a secondary, isolated environment called a Trusted Execution Environment (TEE). This enables further separation from any untrusted code. The capability is typically implemented using secure hardware such as ARM TrustZone® technology.

TEE is responsible for some of the most security-critical operations on the device, including:

1. **Lock screen passcode verification**: available on devices that support a secure lock screen and ship with Android 7.0 or higher. Lock screen verification is provided by the TEE unless an even more secure environment, like tamper-resistant hardware, is available.
2. **Fingerprint template matching**: available on devices that have a fingerprint sensor and

ship with Android Marshmallow 6.0 or higher.
3. **Protection and management of KeyStore keys**: available on devices that support a secure lock screen that ship with Android 7.0 or higher.

Tamper-resistant hardware

Some devices, such as the Google Pixel 2, ship with tamper-resistant hardware to perform some security-critical operations. This hardware is built with additional protections against physical tampering and only shares very limited resources with the main application processor, significantly reducing its attack surface and the potential of side channel attacks. Starting with Android 8.0 Oreo, compatible devices use tamper-resistant hardware to verify the lock screen passcode.

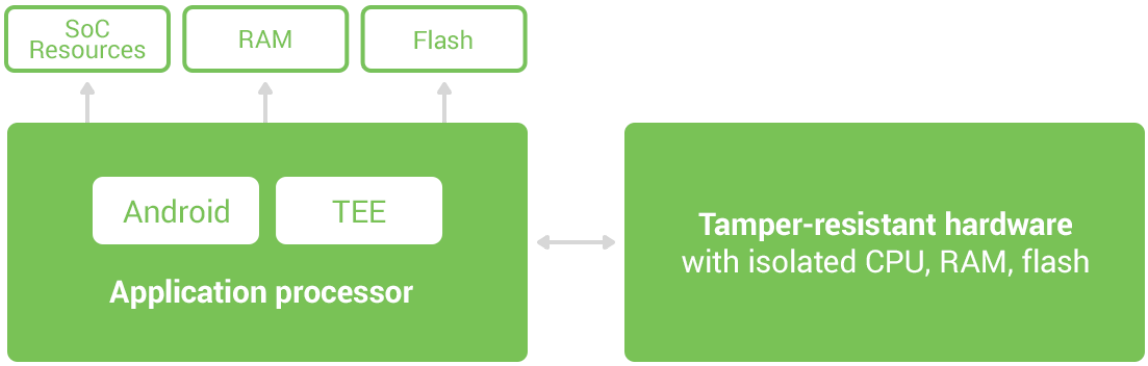


Figure 1. Tamper-resistant hardware creates numerous protections on the device.

Device integrity

Device integrity features protect the mobile device from any changes to the operating system. With companies using mobile devices for essential communication and core productivity tasks, keeping the OS secure is essential. Without device integrity, very few security properties can be assured. Android adopts several measures to guarantee device integrity at all times.

Verified Boot

Verified Boot mitigates attacks against devices by providing a boot process that verifies system software using a hardware root of trust. This makes it more difficult for software attacks to result in a persistent OS compromise, and provides users with a safe state at boot time.

Each Verified Boot stage is cryptographically signed. Each phase of the boot process verifies the integrity of the subsequent phase, prior to executing that code. As of Android 7.0, full boot of a compatible device with a locked bootloader proceeds only if the operating system satisfies integrity checks.

As of Android 8.1, device implementations with more than 1GB of RAM must support Verified Boot for device integrity. Verification algorithms used must be as strong as current recommendations

from NIST for hashing algorithms (SHA-256) and public key sizes (RSA-2048).

The Verified Boot state is used as an input in the process to derive disk encryption keys. If the Verified Boot state changes (e.g. the user unlocks the bootloader), then the secure hardware prevents access to data used to derive disk encryption keys.

Verified Boot on compatible devices running Android 8.0 and higher enables rollback protection. This means that a kernel compromise (or physical attack) cannot put an older, more vulnerable, version of the OS on your system and boot it. Additionally, rollback protection state is also stored in tamper-evident storage.

Enterprises can check the state of Verified Boot using KeyStore key attestation. This retrieves a statement signed by the secure hardware attesting to many attributes of Verified Boot along with other information about the state of the device. KeyStore key attestation is required in devices shipped with Android 8.0 onwards.

Find out more about Verified Boot here.

Sandboxing

Android runs all apps inside sandboxes to prevent malicious or buggy app code from compromising other apps or the rest of the system. Because the application sandbox is enforced in the kernel, this enforcement extends to the entire app regardless of the specific development environment, APIs used, or programming language. A memory corruption error in an app only allows arbitrary code execution in the context of that particular application, with the permissions enforced by the OS.

Similarly, system components run in least-privileged sandboxes in order to prevent compromises in one component from affecting others. For example, externally reachable components, like the media server and WebView, are isolated in their own restricted sandbox.

Android employs several sandboxing techniques, including Security-Enhanced Linux (SELinux), seccomp, and file-system permissions.

SELinux

Android uses SELinux to enforce mandatory access control (MAC) over all processes and apps, even processes running with root and superuser privileges. SELinux provides a centralized auditable security policy that can be used to strongly separate processes from one another.

Android devices implement SELinux policy on a per-domain basis in enforcing mode—no permissive mode domains are allowed. Illegitimate actions that violate policy are blocked and all violations (denials) are logged by the kernel. They are then readable using the dmesg and logcat command-line tools.

As of Android 8.0, with Project Treble, SELinux is used to enforce a separation between the framework and the device-specific vendor components such that they run in different processes and communicate with each other via a set of standard vendor interfaces implemented as Hardware

Abstraction Layers (HALs). Device OEMs can create a HAL implementation that runs in its own sandbox and is only permitted to access the hardware driver it controls, and permissions granted to the process are limited to only those required to do its job. On the framework side, the client runs in a sandbox that does not allow it access to hardware drivers and other permissions and capabilities needed by the HAL implementations.

Seccomp filter

In conjunction with SELinux, Android uses Seccomp to further restrict access to the kernel by blocking access to certain system calls. As of Android 7.0, Seccomp was applied to processes in the media frameworks. As of Android 8.0, a seccomp filter is installed into zygote, the process from which all the Android apps are derived. It blocks access to certain system calls, such as swapon/swapoff, which have been implicated in some security attacks. Additionally, it blocks the key control system calls, which are not useful to apps.

Filesystem sandboxing

Android uses Linux filesystem-based protection to further isolate application resources. Android assigns a unique user ID (UID) to each application and runs it as that user in a separate process. By default, apps cannot access each other's files or resources just as different users on Linux are isolated from each other.

Data protection

Android uses industry-leading security features to protect user data. The platform creates an application environment that protects the confidentiality, integrity, and availability of user data.

Encryption

Encryption is the process of encoding user data on an Android device using an encryption key. With encryption, even if an unauthorized party tries to access the data, they won't be able to read it. Android supports two methods for device encryption: file-based encryption and legacy full-disk encryption.

File-Based Encryption

File-based encryption (FBE) can be used on devices to take full advantage of Direct Boot APIs and offer a user-friendly and secure experience.

Direct Boot allows encrypted devices to boot straight to the lock screen. Previously, on encrypted devices using full-disk encryption (FDE), users needed to provide credentials before using basic phone functionality. For example, alarms could not operate, accessibility services were unavailable, and phones could not receive calls until a user provided credentials.

With file-based encryption and APIs to make apps aware of encryption, it's possible for these apps to operate within a limited context before users have provided their credentials while still protecting

private user information.

On a file-based encryption-enabled device, each device user has two storage locations available to apps:

- Credential Encrypted (CE) storage, which is the default storage location and only available after the user has unlocked the device. CE keys are derived from a combination of user credentials and a hardware secret. It is available after the user has successfully unlocked the device the first time after boot and remains available for active users until the device shuts down, regardless of whether the screen is subsequently locked or not.
- Device Encrypted (DE) storage, which is a storage location available both during Direct Boot mode and after the user has unlocked the device. DE keys are derived from a hardware secret that's only available after the device has performed a successful Verified Boot.

By default, apps do not run during Direct Boot mode. If an app needs to take action during Direct Boot mode, such as an accessibility service like Talkback or an alarm clock app, the app can register components to run during this mode.

DE and CE keys are unique and distinct - no user's CE or DE key will match another. File-based encryption allows files to be encrypted with different keys, which can be unlocked independently. All encryption is based on AES-256 in XTS mode. Due to the way XTS is defined, it needs two 256-bit keys. In effect, both CE and DE keys are 512-bit keys.

By taking advantage of CE, file-based encryption ensures that a user cannot decrypt another user's data. This is an improvement on full-disk encryption where there's only one encryption key, so all users must know the primary user's passcode to decrypt data. Once decrypted, all data is decrypted.

Find out more about File-Based Encryption.

Full-Disk Encryption

Full-disk encryption is the process of encoding all user data on an Android device using a single encryption key. After 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 full-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 OpenSSL library. Some devices may use 256-bit AES.

Upon first boot, the device creates a randomly generated 128-bit master key and then hashes it with a default password and stored salt. This hash is then passed through a keyed function based on RSA in the Trusted Execution Environment, to prevent offline password guessing. When the user updates their passcode, the hash is regenerated without regenerating the master key.

Find out more information about Full-Disk Encryption.

Lock screen

For devices running Android 7.0 or higher, both fingerprint template matching and passcode verification can only take place on secure hardware with rate limiting (exponentially increasing timeouts) enforced.

Tamper-resistant hardware support

As of Android 8.0, compatible devices can optionally use tamper-resistant hardware to verify the lock screen passcode. If verification succeeds, the tamper-resistant hardware returns a high entropy secret that can be used to derive the disk encryption key.

Fingerprint

The Android Compatibility Definition Document specifies the following requirements for all Android devices with a fingerprint sensor:

- A false acceptance rate not higher than 0.002% and a false rejection rate of less than 10%.
- After 5 rejected attempts, leave at least 30 seconds between subsequent attempts.
- A hardware-backed keystore implementation, and fingerprint matching in a Trusted Execution Environment (TEE) or on a chip with a secure channel to the TEE.
- All identifiable fingerprint data must be encrypted and cryptographically authenticated such that they cannot be acquired, read, or altered outside of the TEE.
- Prevent addition of a fingerprint without first establishing a chain of trust by having the user confirm an existing device credential or add a new device credential (PIN/pattern/password).

Android offers APIs that allow apps to use fingerprints for authentication, and allows users to authenticate by using their fingerprint scans on supported devices. These APIs are used in conjunction with the Android Keystore system.

Additional authentication methods and biometrics

Android supports the Trust Agent framework to unlock the device. Google Smart Lock uses that framework to allow a device to remain unlocked as long as it stays with the user, as determined by certain user presence or other signals.

However, note that Smart Lock does not meet the same level of assurance as other unlock methods on Android and is not allowed to unlock auth-bound KeyStore keys. Organizations can disable this using the KEYGUARD_DISABLE_TRUST_AGENTS flag.

Application security

Apps are an integral part of any mobile platform, and users increasingly rely on mobile apps for core productivity and communication tasks. Android provides multiple layers of application protection, enabling users to download apps for work or personal use to their devices with the peace of mind that they're getting a high level of protection from malware, security exploits, and attacks.

Application signing

Android requires that all apps be digitally signed with a developer key prior to installation. Android

uses the corresponding certificate to identify the application's author. When the system installs an update to an application, it compares the certificate in the new version with the one in the existing version, and allows the update if the certificate matches.

Android allows apps signed with the same key to run in the same process, if the apps 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 the same key. By signing multiple apps with the same key, and using signature-based permissions, an app can share code and data in a secure manner.

Android permissions

Apps, by default, have very limited capabilities and must get additional permissions from the user, such as access to contacts and SMS messages. Apps that target API level 23 or higher use runtime permissions, where they prompt users to accept permissions at runtime rather than at installation. This runtime permissions approach streamlines the app install and update process, since the user does not need to grant permissions when they install or update the app. It also gives the user more control over the app's functionality; for example, a user could choose to give a camera app access to the camera. The user can revoke the permissions at any time, by going to the app's Settings screen.

Android 8.0 includes improvements to give users better control over the use of identifiers. Privacy-sensitive device identifiers are either no longer accessible or gated behind a runtime permission.

For example, APIs that access the Wi-Fi MAC address have been removed except on devices that are fully managed by the organization (fully managed device) through the Device Policy Controller (DPC) running in device owner mode. Access to IMEI is now behind the PHONE permission.

On enterprise devices, device management apps can deny permissions on behalf of the user using the setPermissionPolicy() API.

Hardware-backed KeyStore and KeyChain

KeyStore

The Android KeyStore class lets you manage private keys in secure hardware to make them more difficult to extract from the device. It was introduced in Android 4.3 and focuses on apps storing credentials used for authentication, encryption, or signing purposes.

Keystore supports symmetric cryptographic primitives such as AES (Advanced Encryption Standard) and HMAC (Keyed-Hash Message Authentication Code) and asymmetric cryptographic algorithms such as RSA and EC. Access controls are specified during key generation and enforced for the lifetime of the key. Keys can be restricted to be usable only after the user has authenticated, and only for specified purposes or with specified cryptographic parameters. For more information, see the Authorization Tags and Functions pages.

Additionally, version binding binds keys to an operating system and patch level version. This ensures

that an attacker who discovers a weakness in an old version of system or TEE software cannot roll a device back to the vulnerable version and use keys created with the newer version.

For devices that support a secure lock screen and ship with Android 7.0 or higher, KeyStore must be implemented in secure hardware. This guarantees that even in the event of a kernel compromise, KeyStore keys are not extractable from the secure hardware.

KeyStore key attestation

Devices that ship with Android 8.0 and higher support Key Attestation, which empowers a server to gain assurance about the properties of keys, verify that they are signed properly and confirm they're protected. Devices that support Google Play are provisioned at the factory with an attestation key generated by Google. The secure hardware on such devices can sign statements with the provisioned key, which attests to properties of keys protected by the secure hardware, such as the fact that the key was generated and can't leave the secure hardware. Attestation fields include purpose, padding, activate DateTime, and authTimeout. Additionally, key attestation better enables the location of important properties about the device, such as the OS version, patch level, and whether it passed Verified Boot.

Find out more information about verifying hardware-backed keys with Key Attestation.

KeyChain

Android 4.0 introduced the KeyChain class to allow apps to use the system credential storage for private keys and certificate chains. KeyChain is often used by Chrome, Virtual Private Network (VPN) apps, and many enterprise apps to access keys imported by the user or by the mobile device management app.

Whereas the KeyStore is for non-shareable app-specific keys, KeyChain is for keys that are meant to be shared across profiles. For example, your mobile device management agent can import a key that Chrome will use for an enterprise website.

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 apps, by supporting Transport Layer Security (TLS), including TLS v1.0, TLS v1.1, and TLS v1.2.

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-CCM authenticated encryption.

VPN

Android supports securely connecting to an enterprise network using VPN:

- **Always-on VPN**—The VPN can be configured so that apps don't have access to the network until a VPN connection is established, which prevents apps from sending data across other networks.
    - Starting in Android 7.0, Always-on VPN has been extended to support VPN clients that implement VpnService. The system automatically starts that VPN after the device boots. Device owners and profile owners can direct work apps to always connect through a specified VPN. Additionally, users can manually set Always-on VPN clients that implement VpnService methods using **Settings>More>VPN**. The option to enable Always-on VPN from Settings is available only if the VPN client targets API level 24 or higher.
- **Per User VPN**—On multi-user devices, VPNs are applied per Android user, so all network traffic is routed through a VPN without affecting other users on the device. VPNs are applied per work profile, which allows an IT administrator to specify 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 introduced support to facilitate VPN connections on allowed apps and to prevent VPN connections on disallowed apps.

Third-party apps

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

Android 7.0 and higher support Network security configuration, which lets apps easily customize their network security settings in a safe, declarative configuration file without modifying app code. You can configure these settings for specific domains; for example, opting out of cleartext traffic. This helps prevent an app from accidentally regressing due to changes in URLs made by external sources, such as backend servers.

This safe-by-default setting reduces the application attack surface while bringing consistency to the handling of network and file based application data. For more information, see this Android Security Blog.

Certificate handling

As of Android 7.0, all new devices must ship with the same certificate authority store.

Certificate authorities (CA) are a vital component of the public key infrastructure used in establishing secure communication sessions via Transport Layer Security (TLS). Establishing which CAs are trustworthy—and by extension, which digital certificates signed by a given CA are trustworthy—is critical for secure communications over a network.

With Android 7.0, compatible devices trust only the standardized system CAs maintained in AOSP. Apps can also choose to trust user- or admin- added CAs. Trust can be specified across the whole app or only for connections to certain domains.

When device-specific CAs are required, such as a carrier app needing to securely access components of the carrier's infrastructure (e.g.SMS/MMS gateways), these apps can include the private CAs in the components/apps themselves. For more details, see Network Security Configuration.

Find out more about certificate authorities.

Android security updates

Monthly device updates are an important tool to keep Android users safe. Every month, Google publishes Android Security Bulletins to update users, partners, and customers on the latest fixes. These security updates are available for three years from release.

Additionally, security-critical fixes are pushed to all Pixel devices monthly, while Pixel firmware images are released to the Google Developer site. Many OEM partners follow a similar cadence in their security updates. In total, 30 percent more devices received these regular patches in a year-over-year comparison, with a total of one billion devices receiving such updates in 2017. Many OEMs also detail their security bulletins:

- Google
- LG
- Motorola
- Nokia
- Samsung

Users can find out whether they're running a recently patched device with the Security Patch Level, a value indicating the security patch level of a build. It's available through the attestation certificate chain, which contains a root certificate that is signed with the Google attestation root key. It is also visible in the device settings.

Google security services

Google Play Protect

Google Play Protect is a powerful threat detection service that actively monitors a device to protect it, its data, and its apps from malware. The always-on service is built into any device that has Google Play, protecting more than 2 billion devices.

Google Play Protect regularly scans all the apps on a device, including any not installed from the Play Store, for harmful behavior or security risks. If it detects an app containing malware, it notifies the user, who can then uninstall the application. Google Play Protect can also remove malicious apps automatically as part of its prevention initiative and use the information it gathers to improve the detection of Potentially Harmful Applications (PHAs). In addition, the user can opt to have unknown apps sent to Google for better detection information.

Google Play Protect is available on devices enabled with Google Mobile Services. On devices running Android 4.2 or higher, users can opt out of Google Play Protect, although keeping it on is recommended.

An enterprise can further minimize the potential for malware by using the DISALLOW_INSTALL_APPS user restriction to prevent users from installing any apps to their device when fully managed. With DISALLOW_INSTALL_UNKNOWN_SOURCES, an organization can restrict users to only installing apps from system sources such as the Play Store. ENSURE_VERIFY_APPS can disable the ability to turn off app verification through Google Play Protect for fully managed devices or the work profile.

SafetyNet

SafetyNet is a set of Google Play Protect APIs that protects apps against security threats. This series of APIs can mitigate against device tampering, bad URLs, PHAs, and fake users.

The SafetyNet Attestation API provides several tools to determine the security of the Android environment for apps. These APIs analyze the devices that have installed the application. The service attests if the device is known to Google as CTS compatible. The return value indicates to the calling application (for example, an EMM Agent or other enterprise application) whether the device is a known device running a known build. Additionally, the service provides a third party API in Google Play services, using GoogleApiClient, which returns a value indicating whether the device is in the claimed state.

The SafetyNet Safe Browsing API offers services to determine if a URL has been marked as a known threat by Google. SafetyNet implements a client for the Safe Browsing Network Protocol v4 developed by Google. Both the client code and the v4 network protocol were designed to preserve users' privacy and keep battery and bandwidth consumption to a minimum. Enterprises can use this API to take full advantage of Google's Safe Browsing service on Android in the most resource-optimized way, and without implementing its network protocol.

The SafetyNet service also includes the SafetyNet reCAPTCHA API, which protects apps from malicious traffic. This API uses an advanced risk analysis engine to protect apps from spam and other abusive actions. If the service suspects that the user interacting with the app might be a bot instead of a human, it serves a CAPTCHA that a human must solve before the app can continue executing.

The SafetyNet Verify Apps API allows an app to interact programmatically with Google Play Protect, to check whether there are known potentially harmful apps installed. If an app handles sensitive user data, such as financial information, developers should confirm that the current device is protected against malicious apps and is free of apps that may impersonate it or perform other malicious actions. If the security of the device doesn't meet the minimum security posture, developers can disable functionality within the app to reduce the danger to the user.

Google Play app review

Google Play is Google’s app distribution platform for Android. Together with the work of Google Play

Protect, the Play Store has policies in place to protect users from attackers trying to distribute PHAs.

Developers are validated in two stages. They are first reviewed when they create their developer account based on their profile and credit cards. Developers are then reviewed further with additional signals upon app submission.

Before applications become available in Google Play they undergo an application review process to confirm they comply with Google Play policies. Google has developed an automated application risk analyzer that performs static and dynamic analysis of APKs to detect potentially harmful app behavior. When Google's application risk analyzer discovers something suspicious, it flags the offending app, and refers it to a security analyst for manual review.

Google Play Protect also regularly scans Play apps for malware and other vulnerabilities. Google is constantly improving its system tools and methods, applying new machine learning techniques, and updating detection and response systems to protect against new vulnerabilities and PHAs. Additionally, Google suspends developer accounts that violate developer program policies.

Google Play also has ratings and reviews that provide users information about an application before installing it. Apps that aren't forthcoming about their practices tend to have low star ratings and poor comments. The Play Store aggregates review scores and highlights the most useful comments on the application page. Additionally, developers have the opportunity to engage with reviews, building a trusted environment for both developers and Android users.

In 2017, Google Play launched the Google Play Security Rewards program to work with the developer community on identifying security threats to apps, such as Remote Code Execution (RCE). If a researcher discovers an RCE and discloses it to the developer, Google rewards the researcher with a reward, whether or not the developer is enrolled in the rewards program. This has been tremendously successful in assisting with the tracking of potential vulnerabilities.

Another key element in minimizing risk is the use of updated APIs. Encouraging developers to use the most recent APIs nudges them to support the most updated features to create the best security and performance. In the second half of 2018, Play will require that new apps and app updates target a recent Android API level. This is required for new apps in August 2018, and for updates to existing apps in November 2018.

Vulnerability Reward Program

Google offers a rapid response program to handle vulnerabilities found in Android by working with hardware and carrier partners to quickly resolve security issues and push security patches.

Google's Vulnerability Reward Program encourages and rewards researchers who find, fix, and prevent vulnerabilities on Android. The Android Security Rewards program covers security vulnerabilities discovered in the latest available Android versions for Pixel devices and tablets currently for sale in the Google Store. The program rewards reported vulnerabilities rated as Critical, High, and Moderate.

Additionally, Android is also 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), an international coalition of security incident response experts.

Device and profile management

Android supports several different enterprise deployment models:

1. **Bring Your Own Device (BYOD)**: Personal device with work apps in a separate container.
2. **Work-only device:** Corporate liable device that the enterprise owns and fully controls.
3. **Personally enabled work device:** Corporate liable device that allows for personal tasks.
4. **Dedicated device**: Corporate liable device that fulfills a dedicated use case, such as digital signage, ticket printing, or inventory management.

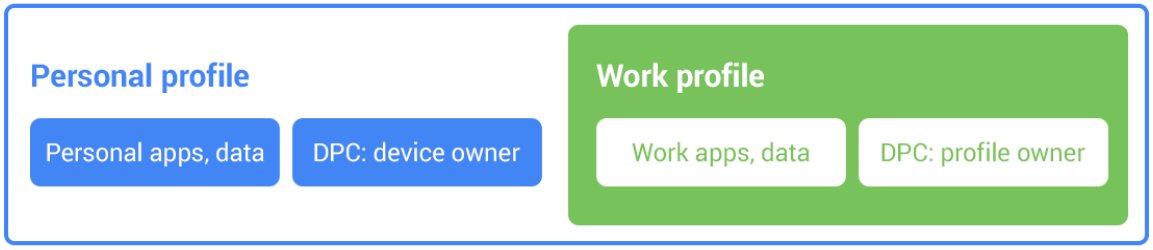
In order to support deployment models where work and personal apps live together on the same device, Android puts them into different containers, or **profiles**, with a firewall between them enforced by the operating system. Personal apps and data are always in the primary profile. Work apps and data are in a second profile called the work profile for BYOD and personally enabled work device deployments. In work-only device and dedicated device modes, only work data is present and stays in the primary profile.

Each profile may be managed by an EMM app acting as a **Device Policy Controller (DPC)**. The DPC has access to the device management APIs available in the DevicePolicyManager class and receives callbacks from the system via the DeviceAdminReceiver class.

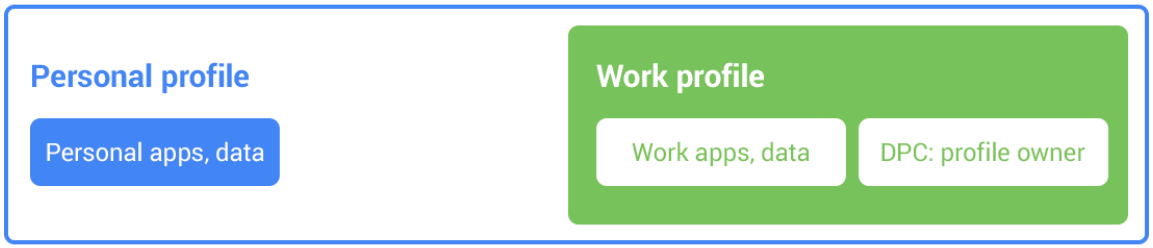
The DPC runs in one of two main modes:

1. **Device Owner**: runs in the primary profile and has the ability to manage a device in fully managed device mode. This is appropriate for corporate liable devices.
2. **Profile Owner**: runs in and manages only the work profile.

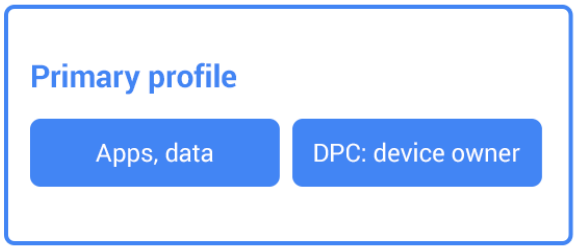
For example, a typical personally enabled work device configuration looks like this:



A typical BYOD configuration would not have a DPC running in the personal profile and looks like the following:



When only work apps and data are present on the device, such as in a typical dedicated device configuration, then only the primary profile exists with a DPC running.



Learn more about the capabilities available to device and profile owners here.

Learn more about work profiles on fully managed devices here.

Device policies

Most capabilities available to the DPC are accessible via the DevicePolicyManager APIs and user restrictions in UserManager.

They can prevent sharing of files from the work profile or device, such as:

1. DISALLOW_BLUETOOTH_SHARING: disallows transferring files via Bluetooth.
2. DISALLOW_USB_FILE_TRANSFER: disallows sending files via USB.
3. DISALLOW_OUTGOING_BEAM: disallows beaming out data from apps using NFC.
4. DISALLOW_MOUNT_PHYSICAL_MEDIA: disallows mounting physical external media.

Device owner and profile owner mode also have a lot of control over other aspects of the device or profile.

1. DISALLOW_DEBUGGING_FEATURES: disallows access to debugging capabilities.
2. DISALLOW_AUTOFILL: disallows autofill services.
3. Setting device passcode policy using APIs such as setPasswordQuality().
4. Disabling less secure unlock methods using setKeyguardDisabledFeatures().
5. Disabling the camera using setCameraDisabled().
6. Setting permitted accessibility services using setPermittedAccessibilityServices().
7. Setting permitted input methods using setPermittedInputMethods().
8. Disabling screen capture using setScreenCaptureDisabled().
9. Automatically accepting/denying some runtime permissions with setPermissionPolicy().

10. If the device is lost, DPC can lock (lockNow()) or wipe (wipeData()) the device.
11. Disable backups using setBackupServiceEnabled().
12. Disallow adding a personal account using DISALLOW_MODIFY_ACCOUNTS. This makes it harder to copy corporate data to personal cloud accounts.
13. Require Google Play Protect to be enabled using ENSURE_VERIFY_APPS.
14. Require only installing apps from known sources such as the Play store using DISALLOW_INSTALL_UNKNOWN_SOURCES.
15. Install keys and certificates into the profile-wide KeyChain using installKeyPair(), and control access to those keys. These can be used as machine certificates to identify the device.
16. Set always on VPN using setAlwaysOnVpnPackage().

As a general guide, DO controls the primary profile and PO controls the work profile. However, there are circumstances whereby PO can enable a global user restriction. For example, the ENSURE_VERIFY_APPS flag enforces app verification across all users on the device when running Android 8.0 and higher. Admins should reference the Android developer documentation to verify the specific implementations of each API.

Google provides an open-source app, Test DPC, for testing enterprise functionality in your DPC app. Test DPC is available from github or Google Play. You can use Test DPC to:

- Simulate features in Android
- Set and enforce policies
- Set app and intent restrictions
- Set up work profiles
- Set up fully managed Android devices

Because the DPC stays enabled throughout the entire lifetime of the device or profile it manages, it can ensure that its policies are always followed.

Device owner mode provisioning

The lifetime of the DPC is always tied to the lifetime of the device or profile it manages. IT managers, or the end user, must enroll a device into fully managed device mode, which provisions the device policy client as a device owner. Provisioning must occur during the initial setup of a new device, or after a factory reset. In the case of DO, it can only be provisioned during initial device setup and only be removed by the DO itself.

A number of options exist to provision a device into fully managed device mode:

- Android zero-touch enrollment - after creating a configuration in the cloud, the IT admin can deploy a device to an end-user who is enrolled into management during setup wizard.
- NFC / QR code - An administrator provisioning large numbers of devices or an employee setting up their own single device can perform an NFC bump or scan a QR code to install the necessary DPC and initiate the enrollment process.
- Google Account - With the Google Account provisioning method, the DPC guides the user through the provisioning steps after the user adds their Google Account during the initial device setup or via settings. An admin receives an EMM token from the Google Admin console, which is used to call the enroll method of the Google Play EMM API.

A device in fully managed device mode can be configured with factory reset protection so the user cannot factory-reset the device to remove the device policy client and turn it into a personal device.

Work profile security

Work profile mode is initiated when the DPC initiates a managed provisioning flow. The work profile is based on the Android multi-user concept, where the work profile functions as a separate Android user segregated from the primary profile. The work profile shares common UI real estate with the primary profile. Apps, notifications, and widgets from the work profile show up next to their counterparts from the primary profile and are always badged so users have an indication as to what type of app it is.

With the work profile, enterprise data does not intermix with personal application data. The work profile has its own apps, its own downloads folder, its own settings, and its own KeyChain. It is encrypted using its own encryption key, and it can have its own passcode to gate access.

The work profile is provisioned upon installation. Administrators can also remotely instruct the device policy client to remove the work profile, for instance, when a user leaves the organization or a device is lost. Whether the user or an IT administrator removes the work profile, user data in the primary profile remains on the device.

Separate work challenge

Android 7.0 introduced support for a separate work challenge to enhance security and control. The work challenge is a separate passcode that protects work apps and data. Admins managing the work profile can choose to set the password policies for the work challenge differently from the policies for other device passwords. Admins managing the work profile set the challenge policies using the usual DevicePolicyManager methods, such as setPasswordQuality() and setPasswordMinimumLength(). These admins can also configure the primary device lock, by using the DevicePolicyManager instance returned by the DevicePolicyManager.getParentProfileInstance() method.

As part of setting up a separate work challenge, users may also elect to enroll fingerprints to unlock the work profile more conveniently. Fingerprints must be enrolled separately from the primary profile as they are not shared across profiles.

As with the primary profile, the work challenge is verified within secure hardware, ensuring that it's difficult to brute-force. The passcode, mixed in with a secret from the secure hardware, is used to derive the disk encryption key for the work profile, which means that an attacker cannot derive the encryption key without either knowing the passcode or breaking the secure hardware.

Cross profile data sharing

While data in the work profile is segregated by default from the user's personal data, there are instances where sharing is useful. Android allows sharing between profiles in ways that can be managed by the DPC. For example:

1. Disallow copy & paste between profiles: DISALLOW_CROSS_PROFILE_COPY_PASTE

2. Allow the primary profile to handle web links from the work profile: ALLOW_PARENT_PROFILE_APP_LINKING
3. Allow widgets from the work profile, such as a calendar widget, to be added on the home screen: addCrossProfileWidgetProvider()
4. Set whether work profile Caller ID is shown in primary profile: setCrossProfileCallerIdDisabled()
5. Set whether work profile contacts are shown in primary profile: setCrossProfileContactsSearchDisabled()
6. Set which apps can see notifications from the work profile: setPermittedCrossProfileNotificationListeners()
7. Set whether apps in the primary profile using the ACTION_SEND intent may share into the work profile using the DISALLOW_SHARE_INTO_MANAGED_PROFILE user restriction available as of Android P DP1. Note that, to reduce the risk of data leakage, the opposite direction is not allowed by default, though it can be enabled by the DPC.

IT administrators can also control cross profile intents using the addCrossProfileIntentFilter and clearCrossProfileIntentFilters methods available in Android 5.0 and higher. By default, during work profile creation the system automatically configures the following intents to be forwarded to the primary profile:

- Telephony intents—as of Android 7.0, admins can also whitelist a dialer for work, which allows a "business" phone account to make and receive work calls instead of forwarding telephony intents to the primary profile dialer. In this case, all calls from the work dialer are inserted into the work call log.
- Mobile network settings.
- Home intent—to invoke the launcher in the primary profile since it 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.

Application management

The EMM app acting as a DPC controls which work apps are installed. On a fully managed device, the EMM app can call the PackageInstaller APIs directly to silently install, uninstall, and update apps. It can also listen for broadcasts such as ACTION_PACKAGE_ADDED, ACTION_PACKAGE_REMOVED, and ACTION_PACKAGE_REPLACED to be notified of changes to installed apps.

On devices that ship with Google Play, an EMM can delegate app management to Google Play. Through managed Google Play, an enterprise version of Google Play, IT administrators can easily find, deploy, and manage work apps while ensuring that malware and other threats are neutralized.

Managed Google Play

Managed Google Play provides APIs to EMM vendors that allow them to manage apps on Android

devices. Installation of apps in either the work profile or on fully managed devices is possible via managed Google Play, either by direct user request in the managed Google Play Store app (pull), or as a result of a call to the EMM API (push). The APIs provide functionality for use (indirectly) by EMM-managed enterprise administrators as follows:

- An IT administrator can remotely install or remove apps on managed Android devices. This action is limited to devices or profiles that are under management by the EMM.
- An IT administrator can define which users see which apps. A user running the Play Store app within the work profile only sees apps whitelisted for them. The user can install these apps, but not others.
- Enterprise administrators can see which users have apps installed or provisioned, and the number of licenses purchased and provisioned.

Private apps

With Managed Google Play, an enterprise customer can publish apps and target them privately (i.e., they're only visible and installable by users within that enterprise). 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, global 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. In all cases, apps must comply with all Google Play policies.

Unknown sources and sideloading

By default, apps are not permitted to install apps from unknown sources. The *install unknown apps* setting under **Apps & notifications > Special App Access** is set to *Not Allowed*. Administrators for fully managed devices or work profile devices 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 the DISALLOW_INSTALL_UNKNOWN_SOURCES user restriction in both device owner and profile owner is false. If set to true, the user cannot modify the Unknown sources security setting on the device (when called by device owner) or work profile (if called by profile owner).

Additionally, the sideloading of apps using Android Debug Bridge (adb) can be disabled via the DISALLOW_DEBUGGING_FEATURES user restriction in a fully managed device by device owner or work profile by profile owner. The default value of DISALLOW_DEBUGGING_FEATURES for both fully managed device and work profile is false.

In a corporate-managed device or profile, when EMMs set DISALLOW_INSTALL_UNKNOWN_SOURCES and DISALLOW_DEBUGGING_FEATURES user restrictions

to true, IT administrators receive an extra measure of assurance that only company-approved apps will be deployed using managed Google Play to users in a corporate-managed device or profile.

Managed app configuration

Android 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 **Google Play EMM API** allows EMMs to configure these remotely.

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.

Android Enterprise Recommended

Google recently launched Android Enterprise Recommended, an initiative that sets an elevated standard for enterprise devices and services.

Devices in the program meet a set of specifications for hardware, deployment, security updates, and user experience. In addition, OEMs in the program receive an enhanced level of technical support and training from Google. Organizations can then select devices from the curated list with confidence that they meet a common set of criteria, required for inclusion in the Android Enterprise Recommended program:

- Minimum hardware specifications for Android 7.0+ devices
- Support for bulk deployment of Android devices including zero-touch enrollment
- Mandatory delivery of Android security updates within 90 days of release from Google (30 days recommended), for a minimum of three years
- Availability of unlocked devices, to support global deployments
- Minimal preloaded software on managed devices

Conclusion

Security is often associated with a closed ecosystem. Yet the open source development approach of Android is a key part of its security. Developers, device manufacturers, security researchers, SoC vendors, and the wider Android community creates a collective intelligence that locates and mitigates vulnerabilities for the entire ecosystem.

With Android, multiple layers of security support the diverse use cases of an open platform while also enabling sufficient safeguards to protect user and corporate data. Additionally, Android platform security keeps devices, data, and apps safe through tools like app sandboxing, exploit mitigation and device encryption. A broad range of management APIs gives IT departments the tools to help prevent data leakage and enforce compliance in a variety of scenarios. The work profile enables enterprises to create a separate, secure container on users' devices where apps and critical company data are kept secure and separate from personal information.

Google Play Protect, the world's most widely deployed mobile threat protection service, delivers built-in protection on every device. Powered by Google machine learning, it's always working to catch and block harmful apps and scan the device to root out any vulnerabilities. Safe Browsing protection in Chrome protects enterprise users as they navigate the web by warning of potentially harmful sites.

Enterprises rely on smart devices for critical business operations, collaboration, and accessing proprietary data and information. Google continues to invest in resources to make Android a secure platform, and we look forward to further contributions from the community and seeing how organizations will use Android to drive business success.