Name: Mazen Amr ID: 20156011

Name: Hisham Mahfouz ID:20156015

Security Support in Android Platform

Android System Structure

● An open source OS that’s built on the Linux kernel

● Provides an environment for multiple apps to run simultaneously

● Apps are signed and isolated into application sandboxes

● sandbox defines the privileges available to the application

● Apps are generally built using Android Runtime

● Apps interact with the OS through a framework

● The Framework describes system services, APIs and message formats

● To achieve the above properties Android utilizes the Following.

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.

**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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.