DISPLAY TEN

DisplayTen Network Security White Paper

July 2016

DISPLAY TEN

Table of Contents

Introduction 3

1. Cryptography and Data Protection 4-6
2. Application security 7-11
3. Network security 12-13
4. Security best practices 14

Introduction

About this document

This white paper provides an overview of various security features that are in place—at the OS level in our interactive displays.

About DisplayTen:

DisplayTen delivers turnkey solutions to simplify collaboration for the next generation workplaces. DisplayTen's cable-free collaboration solution allows users to share content on the 55", 70" and 84" interactive displays. It features a smart pen and eraser, two-way control between DisplayTen and your Mac/PC, and an elegant modular design.

Cryptography and Data Protection

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

- 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 forceencryptflag 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 /dataand 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 is BoundKeyAlgorithm 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 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 Android security white paper 7 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.

DISPLAY TEN

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.

Security Best Practices

We have designed this 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:
● Strives to preventsecurity 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 minimizethe impact of security issues with features like the application sandbox
● Detectsvulnerabilities 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).