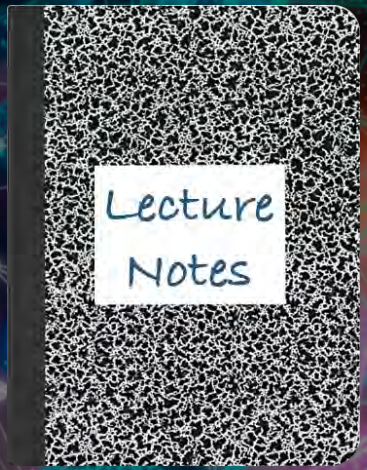
CS 419: Computer Security

Week 13: Part 3
Mobile Device Security



Paul Krzyzanowski

Threat Landscape

Mobile Devices: Users

- **Lots of users!**
  - 3.6B Android users, ~1.46B iOS users
  - Much of the world is mobile-first
- **Users don't think of phones as computers**
  - Social engineering may work more easily on phones
- **Small form factor**
  - Users may miss security indicators (e.g., certificates on web sites)
  - Easy to lose/steal a device
- **Users tend to pick bad PINs**
- **Users may grant app permission requests without thinking**

U.S.: Android: 48.5%, iOS: 51.2%
Worldwide: Android: 70.7%, iOS: 28.6%

https://gs.statcounter.com/os-market-share/mobile/worldwide

https://9to5mac.com/2020/01/28/apple-hits-1-5-billion-active-devices-with-80-of-recent-iphones-and-ipads-running-ios-13/

Mobile Devices: Interfaces

- **Phones have lots of sensors**
  - GSM/3G/4G LTE/5G – Wi-Fi – Bluetooth – GPS – NFC – Microphone
  - Cameras – 6-axis Gyroscope and Accelerometer – Barometer
  - Magnetometer (compass) – Proximity – Ambient light – LiDAR
  - Fingerprint – Face

- **Sensors enable attackers to monitor the world around you**
  - Where you are & whether you are moving
  - Conversations
  - Video
  - Sensing vibrations due to neighboring keyboard activity led to a word recovery rate of 80%

Mobile Devices: Apps

- **Lots of apps**
  - 2.43 million Android apps on Google Play
  - 2.29 million iOS apps on the Apple App Store
- **Most written by untrusted parties**
  - We'd be wary of downloading these on our PCs
  - With mobile apps we rely on
    - Testing & approval by Google (automated) and Apple (automated + manual)
    - App sandboxing
    - Explicit granting of permissions for resource access
- **Apps often ask for more permissions than they use**
  - Most users ignore permission screens
- **Most apps do not get security updates**

2022 data: https://buildfire.com/app-statistics/

Mobile platforms aren't impervious

Mobile Devices: Platform

- **Mobile phones are comparable to desktop systems in complexity**
  - The OS & libraries will have bugs
- **Single user environment**
- **Limited screen space**
  - No hovering, no multiple browser windows (usually)
- **Malicious apps may be able to get root privileges**
  - Attackers may install rootkits, enabling long-term control while concealing their presence

Some apps are preinstalled (Android)

Malware Found Pre-Installed on Low-Cost Android Smartphones

Data Breach Today
Prevention. Response. Notification. TODAY

Phones Sold Through U.S. Government-Subsidized Program

Prajeet Nair – July 10, 2020

For the second time this year, security researchers have found malware embedded in low-cost Android smartphones distributed through a U.S. government program, security firm Malwarebytes reports.

In this latest case, Malwarebytes analysts found the malware embedded in the "settings" feature of the Android smartphone making it nearly impossible to detect or remove from the devices, according to a new research report.

Malwarebytes obtained an infected ANS UL40 smartphone and studied the malware embedded in the device, according to the report.

https://www.databreachtoday.com/malware-found-pre-installed-on-low-cost-android-smartphones-a-14594

Behind the Scenes: How Criminal Enterprises Pre-infect Millions of Mobile Devices

Fyodor Yarochkin, Zhengyu Dong, Vladimir Kropotov, Paul Pajares • May 11, 2023

Mobile phones may come pre-infected with malicious firmware before they are even delivered to the customers. This is a growing problem for regular users and enterprises. Many businesses produce mobile devices by outsourcing the manufacturing process. The process comes with risks. The supply chain of the outsourced manufacturing can be easily infiltrated by third-party threat actors.

In this presentation, we will dive into the criminal operations of a criminal enterprise that targets mobile phones. The criminal group has infected millions of android devices, mainly mobile phones, but also smart watches, smart TVs and more. The infection turns these devices into mobile proxies, tools for stealing and selling SMS messages, social media and online messaging accounts and monetization via advertisements and click fraud.

Our data shows that this is a continuously growing problem. We manually analyzed dozens of the stock-firmware images to confirm the presence of malicious software in these models. Further, through our telemetry data, we confirmed that there are millions of infected devices operated globally. The main cluster of these devices is in South-East Asia and Eastern Europe, however, this is a truly global problem. URL

Example: fake Facebook authentication

- **July 2020**
  - 25 apps discovered in the Google Play Store that trick Facebook users to give authentication credentials
- **Apps infect target phones with malware**
  - Detects opening of the Facebook app
  - Launches a browser and navigates to Facebook's login window
  - Malware uses JavaScript to copy login credentials and send them to a server

https://www.evina.com/they-steal-your-facebook/

Thousands of Android users downloaded this password-stealing malware disguised as anti-virus from Google Play

Users looking to protect their smartphone from hackers found their devices infected with Sharkbot malware.

Danny Palmer • April 7, 2022

Six phony anti-virus apps have been removed from the Google Play app store because instead of protecting users from cyber criminals, they were actually being used to deliver malware to steal passwords, bank details and other personal information from Android users.

The malware apps have been detailed by cybersecurity researchers at Check Point, who say they were downloaded from Google's official app marketplace by over 15,000 users who were looking to protect their devices, which instead became infected with Sharkbot Android malware.

Sharkbot is designed to steal usernames and passwords, which is does by luring victims into entering their credentials in overlayed windows which sends the information back to the attackers, who can use it to gain access to emails, social media, online banking accounts and more.

https://www.zdnet.com/article/these-android-users-wanted-to-protect-their-phones-from-hackers-instead-they-downloaded-malware/

Ways to Infiltrate an iOS Device

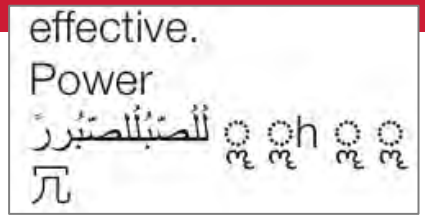
Here are a few ways to get malware onto an iOS device, along with examples of real exploits that used that method.

https://www.theregister.co.uk/2017/07/20/ios_security_skycure/

Threats

- **Privacy**
  - Data leakage
  - Identifier leakage
  - Location privacy
  - Microphone/camera access
- **Security vulnerabilities**
  - Bugs
  - Phishing
  - Malware
  - Malicious Android intents (inter-app communication)
  - Broad access to resources (more than the app needs)

iOS input validation vulnerabilities in Messages



Bugs can surface in unexpected places … over & over

- **May 2015:** "Unicode of Death"
  - Single string in a text message could crash an iPhone
- **Again in Jan 2018:** "*ChaiOS*"
  - Receiving a link causes the messages app to go blank & crash instantly after opening
  - Malformatted characters in the message causes the Webkit HTML engine to crash
  - The target file contains multiple such characters, so CoreText spends a lot of CPU time trying to match fonts for them
- **Again in Feb 2018**
  - A character in an Indian language (Telugu) causes Apple's iOS Springboard to crash when the message is received
  - Messages will no longer open as it fails to load the character
  - Affects third-party messaging apps too

- **Again in May 2018:** Black dot of death
  - Thousands-character-long string of invisible Unicode text causes iMessages to crash when the user launches the app
- **Again in April 2020:** Sindhi characters
  - Several characters from the Sindhi language that cause iOS to lock up and an iPhone to crash

Another data validation problem…

Wallpaper crash explained: Here's how a simple image can soft-brick phones

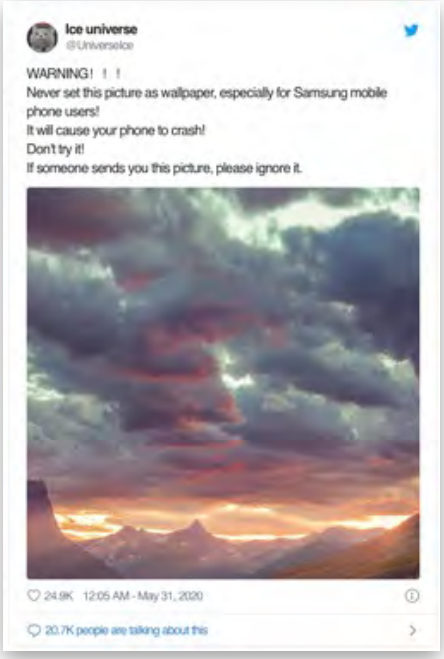
Bogdan Petrovan • June 1, 2020

How can a simple image crash an Android phone to the point that it becomes unusable?

…

Here's a recap: Setting a particular image as wallpaper can send some phones into a loop of crashes that makes them unusable.

There are a few solutions, depending on how hard the phone is hit. Some users were able to change the wallpaper in the short interval between crashes. Others had success deleting the wallpaper using the recovery tool TWRP. But in most cases, the only solution was to reset the phone to factory settings, losing any data that's not backed up.



https://www.androidauthority.com/android-wallpaper-crash-1124577/

4-year campaign backdoored iPhones using possibly the most advanced exploit ever

WIRED

"Triangulation" infected dozens of iPhones belonging to employees of Moscow-based Kaspersky.

Dan Goodin • December 27, 2023

Researchers on Wednesday presented intriguing new findings surrounding an attack that over four years backdoored dozens if not thousands of iPhones, many of which belonged to employees of Moscow-based security firm Kaspersky. Chief among the discoveries: the unknown attackers were able to achieve an unprecedented level of access by exploiting a vulnerability in an undocumented hardware feature that few if anyone outside of Apple and chip suppliers such as ARM Holdings knew of.

“The exploit's sophistication and the feature's obscurity suggest the attackers had advanced technical capabilities,” Kaspersky researcher Boris Larin wrote in an email. “Our analysis hasn't revealed how they became aware of this feature, but we're exploring all possibilities, including accidental disclosure in past firmware or source code releases. They may also have stumbled upon it through hardware reverse engineering.”

…

Over a span of at least four years, Kaspersky said, the infections were delivered in iMessage texts that installed malware through a complex exploit chain without requiring the receiver to take any action. With that, the devices were infected with full-featured spyware that, among other things, transmitted microphone recordings, photos, geolocation, and other sensitive data to attacker-controlled servers. Although infections didn’t survive a reboot, the unknown attackers kept their campaign alive simply by sending devices a new malicious iMessage text shortly after devices were restarted.

https://arstechnica.com/security/2023/12/exploit-used-in-mass-iphone-infection-campaign-targeted-secret-hardware-feature/

Critical bug could have let hackers commandeer millions of Android devices

ars technica

Flaw could be exploited with malicious audio file.

Dan Goodin • April 21, 2022

TextSecurity researchers said they uncovered a vulnerability that could have allowed hackers to commandeer millions of Android devices equipped with mobile chipsets made by Qualcomm and MediaTek.

The vulnerability resided in ALAC—short for Apple Lossless Audio Codec and also known as Apple Lossless—which is an audio format introduced by Apple in 2004 to deliver lossless audio over the Internet. While Apple has updated its proprietary version of the decoder to fix security vulnerabilities over the years, an open-source version used by Qualcomm and MediaTek had not been updated since 2011.

Together, Qualcomm and MediaTek supply mobile chipsets for an estimated 95 percent of US Android devices.

The buggy ALAC code contained an out-of-bounds vulnerability, meaning it retrieved data from outside the limits of allocated memory. Hackers could exploit this mistake to force the decoder to execute malicious code that otherwise would be off-limits.

https://arstechnica.com/information-technology/2022/04/critical-bug-could-have-let-hackers-commandeer-millions-of-android-devices/

This Critical Android Security Threat Could Affect More Than 1 Billion Devices: What You Need To Know

Forbes

Davey Winder • May 26, 2020

Another week, another critical security warning. Here's what most Android users need to know about StrandHogg 2.0

…

The risk being that, if exploited by an attacker, this vulnerability could lead to an elevation of privilege and give that hacker access to bank accounts, cameras, photos, messages and login credentials, according to the researchers who uncovered it. What's more, it could do this by assuming "the identity of legitimate apps while also remaining completely hidden."

…

Rather than exploit the same TaskAffinity control setting as the original StrandHogg vulnerability, StrandHogg 2.0 doesn't leave behind any markers that can be traced. Instead, it uses a process of "reflection," which allows it to impersonate a legitimate app by using an overlay into which the user actually enters credentials. But that's not all; it also remains entirely hidden in the background while hijacking legitimate app permissions to gain access to SMS messages, photos, phone conversations, and even track GPS location details.

https://www.forbes.com/sites/daveywinder/2020/05/26/critical-android-data-stealing-security-threat-confirmed-for-almost-all-android-versions-strandhogg-google-update-warning/?sh=175793d82c82

Android Security

Application Needs

Integrity	App shouldn't be modified between creation & installation
Isolation	Each app needs private data and be protected from other apps
Sharing	Access to shared storage and devices, including the network
Inter-app services	Send messages to other apps to invoke services – when allowed
Portability	Apps should run on different hardware architectures

App Sandboxing

- **Android is built on SELinux**
- **Apps are isolated and can only access their resources**
- **Sandboxing enforced by Linux**
  - Android is a single-user system
  - Each app
    - is assigned a unique user ID on installation
    - runs as a separate process
    - has a private data folder

Core Android services also have their own user IDs

1001	Telephony
1002	Bluetooth
1003	Graphics
1004	Input devices
1005	Audio
etc.	

App Sandboxing: file permissions

Two mechanisms are used

1. **Linux file permissions (discretionary access control)**
   - Owner & root can change permissions
   - Allows an app to share a data file
2. **SELinux mandatory access control**
   - Various data & cache directories
   - Owner cannot change access permissions

Storage

- **External storage:** Shared among all apps
- **Internal storage:** Per-app private directory

App = set of components

- **PC apps have a single launching point & run as a monolithic process**
- **Android apps contain multiple components**
  - Activities
  - Services
  - Broadcast receivers
  - Content providers
- **Example: take and share a photo in Instagram**
  - Instagram requests the camera app
  - Android OS launches the camera app to handle the request
    - User now sees the camera app and no longer interacts with the Instagram app
  - Camera app may access other services, such as a file chooser, that will launch another app
  - User exits the launched app(s), returns to Instagram, and shares the photo

Permissions

Apps need permissions to access services

- System resources: logs, battery levels, …
- System interfaces: Internet, Bluetooth, send SMS, send email, …
- Sensitive data: SMS messages, contacts, email, …
- App-defined services

Services are assigned protection levels:

Normal	Default - no danger to users or system
Dangerous	Access that can compromise the system or privacy – user has to approve during installation or runtime
Signature	Access granted if the app signed by the same developer & contains the same certificate
SignatureOrSystem	Like signature but access granted if a system application is requesting it

Intents

Android apps communicate with system services, between app components, and with other apps via intents

- **Intent = messaging objects**
  {action, data to act on, component to handle the intent} used to
  - Start a service (background) • Start an activity (user-facing & foreground) • Deliver notifications (broadcasts)
- **Explicit intents**
  - App identifies the target component in the intent
- **Implicit intents**
  - App asks Android to find a component based on its data
    E.g., view a web page

App registers its available intents when it is installed

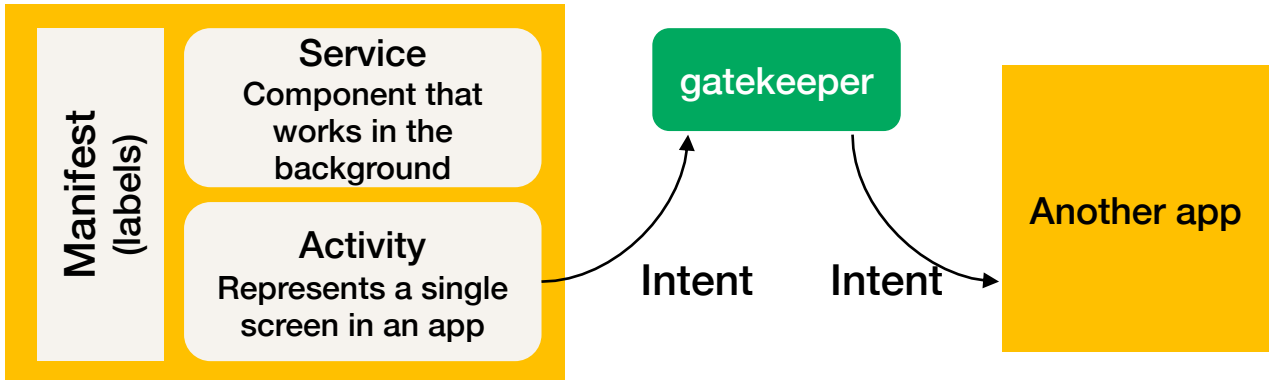
- If several apps register the same intent, the user selects which should be used (e.g., multiple browsers)

Intents vs. Permissions

- **Intents declare app capabilities**
  - Associate actions with components & how they are started
- **Permissions**
  - Identify whether one app is allowed to access another app's component
- **Access to components passes through the gatekeeper, which validates requests**

Intents = *mechanism*

Permissions = *policy*



Platform Security

- **Android creates a read-only partition for the kernel, system libraries, and the app framework**
  - Even malicious apps with elevated privileges cannot modify.
- **Root privileges**
  - Powerful – but not granted to apps
- **Verified boot**
  - Each stage is signed and verified

File Encryption

- **File-based encryption**
  - Device Encrypted (DE) storage: available during boot and after a user unlocked the device
  - Credential Encrypted (CE) storage: available only after the user unlocked the device
- **Uses Linux ext4 file system or F2FS**
  - 512-bit AES file encryption keys
  - Stored in the Trusted Execution Environment (TEE)
    - A separate part of the processor with protected memory running its own OS (Trusty) and communicates with the Linux kernel only via a well-defined messaging API

App Integrity

- **Signed applications**
  - Apps must be signed. Signature validated by Google Play & package manager on the device
- **App verification**
  - Users can enable "verify apps" to have apps evaluated by an app verifier prior to installation
  - Will scan app against Google's database of apps

Exploit Prevention

Android code & Linux execution environment uses

- **Stack canaries**
- **Some heap overflow protections** – check backward & forward pointers
- **ASLR** (Address Space Layout Randomization)
- **No-execute (NX)** hardware protection to prevent code execution on the heap or stack

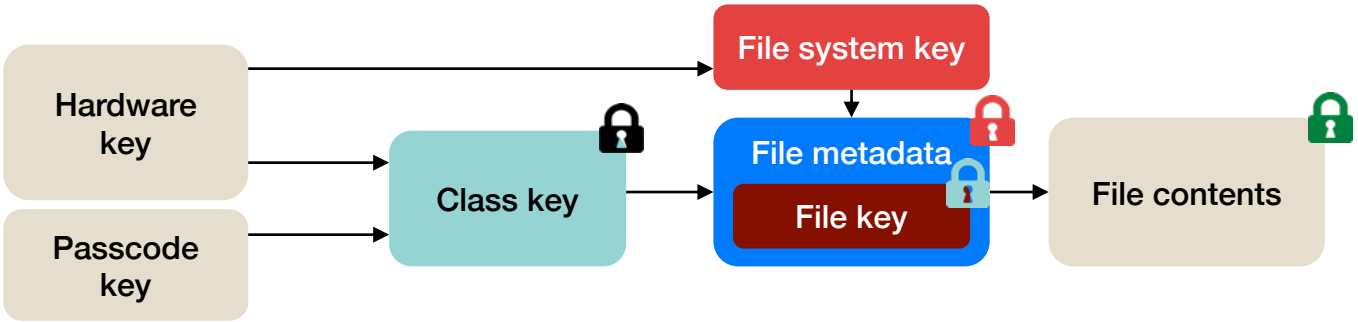
iOS Security

Boot process

- **Boot ROM – trusted read-only component**
  - Contains boot code & Apple root certificate
- **Boot ROM**
  ⇒ Loads Low-Level Boot Loader (LLB)
  ⇒ loads iBoot
  ⇒ loads iOS kernel
- **Each step verifies the signature of the next package**

Data security: encrypted file system & encrypted files

- **Each file gets a random 256-bit AES file key when it is created**
- **File key is encrypted with a class key**
  - Class = user or group; depends on who should access the file
- **Metadata contains file key and info on the class that protects it**
  - Can also specify per-extent keys: portions of a file can be given different keys
- **File metadata is decrypted with the file system key**
  - File system key = random key created when iOS is installed



App Sandbox

All third-party apps are sandboxed

- **Apple kernel-level sandbox**
  - Per-app policy definition file
  - Enforcement of system calls and parameters
  - Controls access to files, system hardware, network
- **Each app has a unique home directory for its files**
  - Restricted from accessing files stored by other apps or making changes to the device
- **System files and resources are shielded from the user's apps**
- **Apps run as a non-privileged user "mobile"**

iOS App Security

- **Runtime protection**
  - System resources & most syscalls not directly accessible by user apps – *must use iOS APIs*
  - App sandbox restricts access to other app's data & resources
  - Inter-app communication through iOS APIs
  - Entire OS partition is read-only
  - Code generation prevented – memory pages cannot be made executable
- **Mandatory code signing**
  - Must be signed using an Apple Developer certificate
  - Root certificate is stored in the hardware – used to validate OS updates
  - iOS performs runtime code signature checks of all executable memory pages

Apps: Entitlements & Extensions

- **Entitlements**
  - Signed key-value pairs that are granted to an app to allow access to services
- **Extensions**
  - Executable binaries packaged within an app that provide functions to other apps
  - Sandboxed and run in their own address space
  - Entitlements restrict extension availability to specific apps

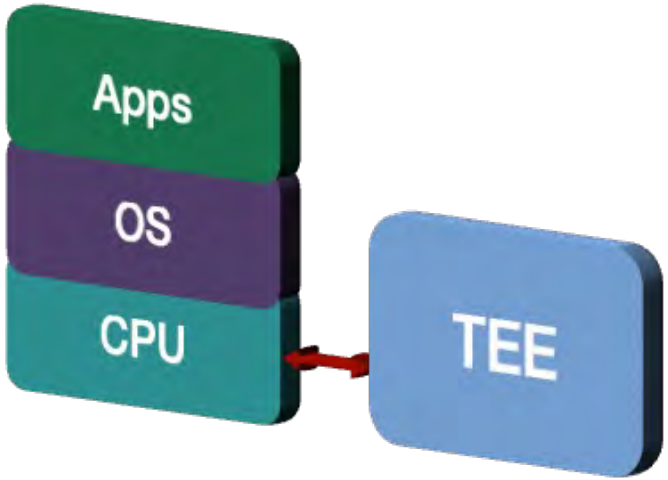
Runtime environment

- Signature checking of each page loaded from an app while executing
- Address space layout randomization (ASLR)
- Memory execute protection – ARM's Execute Never (XN) feature
- Stack canaries

Trusted Execution Environment (TEE)

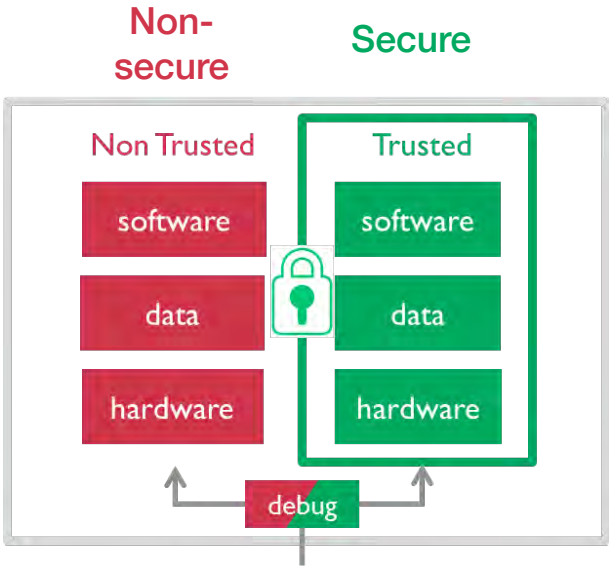
Keeping keys secret

- **For security, we rely on**
  - Keys
  - Encryption, decryption services
  - Signing services
  - Biometric data
- **This data needs to be kept secure**
  - Along with other data, like payment credentials
- **Trusted Execution Environment**
  - Isolated environment for data storage & trusted services



Hardware aids to security: ARM TrustZone

- **Hardware-isolated secure & non-secure worlds**
  - Each CPU core has two virtual cores: secure & non-secure
- **Processor executes in one world at any given time**
- **Each world has its own OS & applications**
- **Software resides in the secure or non-secure world**
  - Non-secure (non-trusted) applications cannot access secure resources directly
- **Proof of device: private key stored in trusted area**
- **Applications**
  - Secure key management & key generation
  - Secure boot, digital rights management, secure payment



http://www.arm.com/products/security-on-arm/trustzone

Android Trusty Trusted Execution Environment

- **Trusty TEE**
  - Isolated from the rest of Android via hardware & software
  - Trusty runs on the same processor as the Android Linux kernel
  - Intel hardware: uses Intel's Virtualization Technology
  - Hardware support
    - ARM: Trustzone™
    - Intel Virtualization Technology
- **Trusty contains**
  - OS kernel derived from Little Kernel (https://github.com/littlekernel/lk)
  - Linux driver to transfer data between the secure Trusty environment & Android
  - Android userspace library to communicate with trusted applications

Android Trusty Trusted Execution Environment

- **Processes in Trusty**
  - Each process runs in unprivileged mode and is isolated from others via memory management
  - All applications are developed by a single party and packaged with the Trusty kernel image, which is signed
  - Verified by the bootloader during boot
- **Uses for Trusty**
  - Gatekeeper subsystem
    - Enrolls and verifies passwords via an HMAC
  - DRM framework for protected content
    - TEE stores device-specific keys needed to decrypt content
    - Main processor sees only the encrypted content and not the keys
  - Mobile payments

iOS: Apple Secure Enclave

Similar to TrustZone but a *separate processor*

- Coprocessor in Apple A7 and later processors: iPhones, TV, Watch, HomePods, Macs
- Runs its own OS (modified L4 microkernel)
- Has its own secure boot & custom software update
- Provides
  - All cryptographic operations for data protection & key management
  - Random number & key generation
  - Secure key store (including UID & GID keys)
  - Biometric auth, including Touch ID (fingerprint) and the neural network for Face ID
  - Authentication for secure payments
- Maintains integrity of data protection even if kernel has been compromised
- Uses encrypted memory
- Communicates with the main processor by an interrupt-driven mailbox and shared memory buffers

Summary

- **Mobile devices are attractive targets**
  - Huge adoption, simple app installation by users, always with the user
- **Android security model**
  - Isolated processes with separate UID and separate VM
  - Java/Kotlin code (mostly, but also native): managed, no buffer overflows
  - Permission model & communication via intents
- **iOS security model**
  - App sandbox based on file isolation
  - File encryption
  - Apps written in Swift (older ones in Objective C)
  - Vendor-signed code, closed marketplace (App Store only)
- **Protection efforts have generally been good**
  - Usually far better than on normal computers … but often not good enough!

The End