ANDROID SECURITY

EMBEDDED SYSTEM PROGRAMMING 2015/2016

PAOLO MONTESEL

BACKGROUND

- Security is hard
- Android was designed with a multi-layered security architecture
- Android has had many security problems in the past and still has
- Google is actively working on it
- End-user actions are also considered in order to mitigate Social Engineering attacks
- Vendor-specific flavours of the OS can introduce new bugs and delay security patches

SECURITY GOALS

- Protect user data
- Protect system resources
- Provide application isolation

OVERVIEW

SECURITY PROGRAM

The entire development lifecycle of Android is subjected to a rigid security program:

- **Design Review**: Each major feature is reviewed by engineers and security experts
- **Penetration Testing/Code Review**: OS components are subject to security reviews both from Google and 3rd party consultants.
- **Community Review**: AOSP code is open and can be reviewed by anyone
- **Incident Response**: Google has a dedicated team in charge of providing rapid mitigation and minimize risk when a bug is reported.

SECURITY MECHANISMS

- Security at the OS level through the Linux kernel
- Mandatory application sandboxing
- Secure IPC
- Application signing
- Application-defined, user-granted permissions

BRIEF ANDROID SECURITY HISTORY

- **1.5**: ProPolice and buffer/integer overflow protections
- **2.3**: Format String protections, No eXecute (NX) bit
- **3.0**: Full Disk Encryption
- **4.0**: Address Space Layout Randomization (ASLR), secure credentials storage
- **4.1**: Position Independent Executables (PIE) support
- **4.3**: SELinux (permissive mode), no more setuid/setgid programs, hardware-backed secure credentials storage
- **4.4**: SELinux (enforcing mode), Certificate Pinning
- **5.0**: Better Full Disk Encryption, encryption by default, non-PIE support dropped
- **6.0**: Runtime permissions, verified boot, Fingerprints

ANDROID SOFTWARE STACK

ANDROID SOFTWARE STACK

KEY CONCEPTS

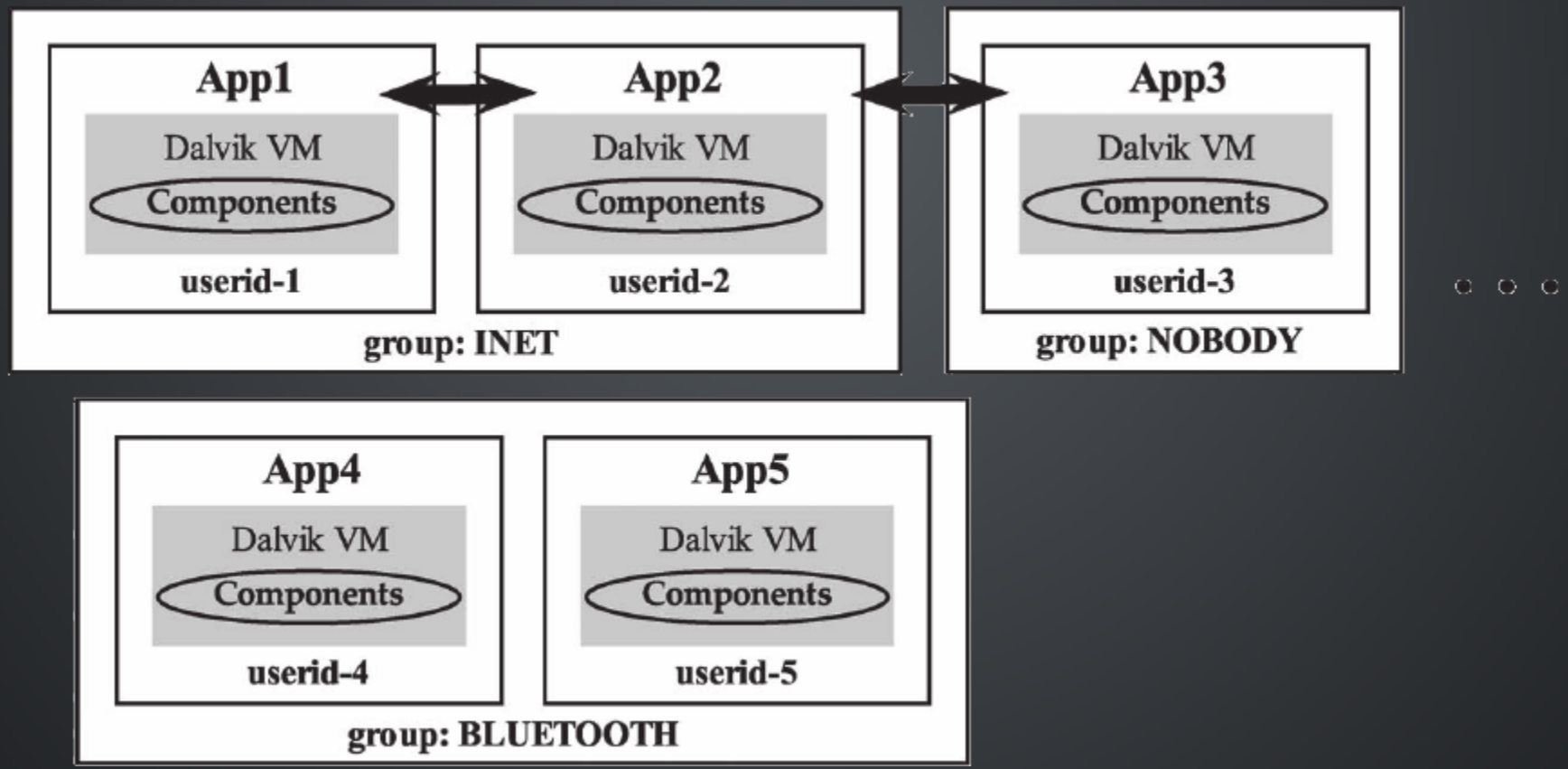
- Each layer in the software stack assumes that the components below are properly secured
- All code above the Linux Kernel is restricted by the Application Sandbox (excluding a small amount of Android code running as root)
- **Device Hardware:** Android is processor-agnostic but takes advantage of hardware-specific security features
- **Android OS**: Built on top of Linux. Device resources are all accessed through the OS
- **Android App Runtime**: Both Dalvik and native applications run in the Application Sandbox

APPLICATION SANDBOX

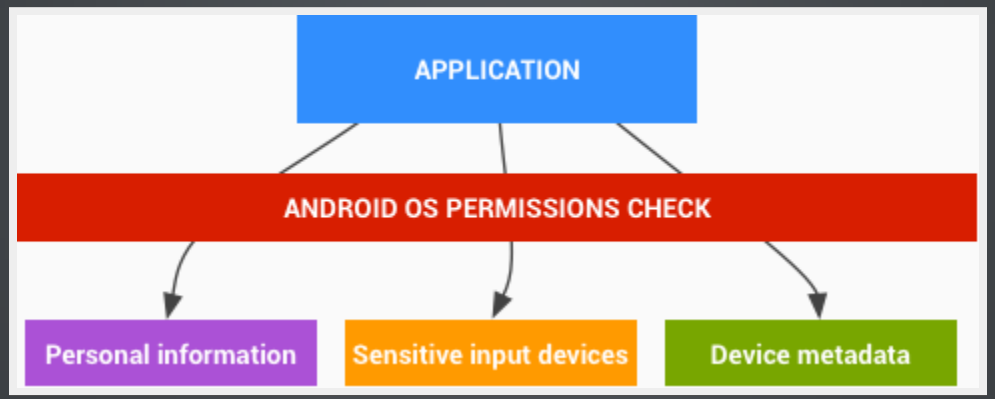
- Kernel-level sandbox
- Each application has a unique user ID
- Each application runs on a separate process
- Apps have limited access to the OS by default
- Apps cannot interact with each other by default
- Native code as secure as the Dalvik code
- To break out of the Sandbox, an attacker must often compromise the Linux Kernel

APPLICATION SANDBOX

- If an app requires a permission, it is assigned the corresponding group ID
- If two App's certificates match, they can share the same UID



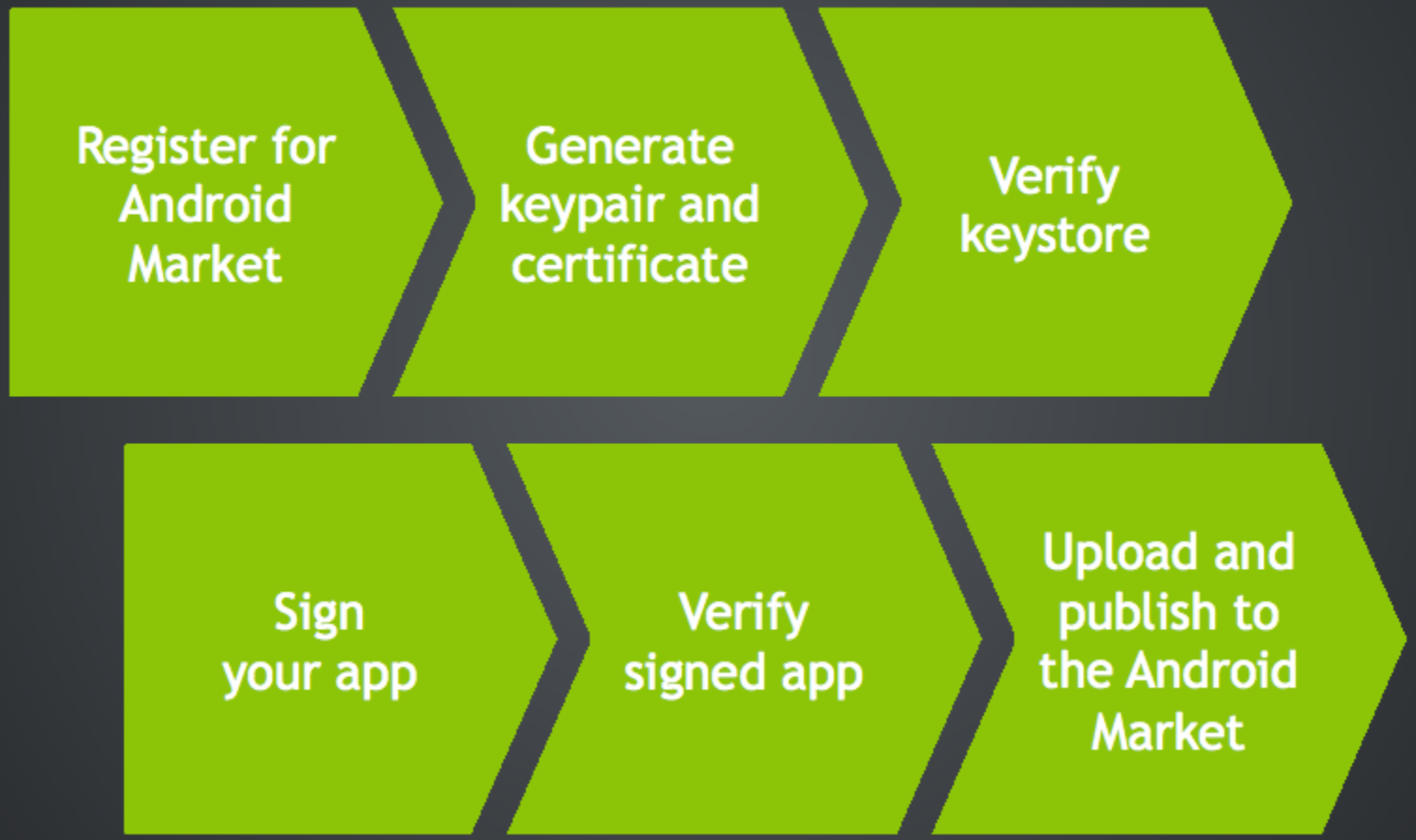
PERMISSION MODEL



PERMISSION MODEL

- By default, Apps can access a limited range of system resources
- Certain APIs are missing on purpose (e.g.: direct SIM-card manipulation APIs)
- Sensitive APIs are protected through a permission mechanism
  - Declared in the *AndroidManifest.xml*
  - Accepted by the user at install-time
  - Requested at run-time from Android 6.0 onwards
- Special treatment is given to cost-sensitive APIs like SMS

APPLICATION SIGNING



APPLICATION SIGNING

- Identifies the author of an App
- Allows for automatically updating Apps in a secure manner
- Proves the integrity of the APK
- If two APKs' signatures match, they can choose to use the same UID
- Since Android 4.2, Apps are verified by default and the OS can block the installation of harmful APKs

AUTHENTICATION

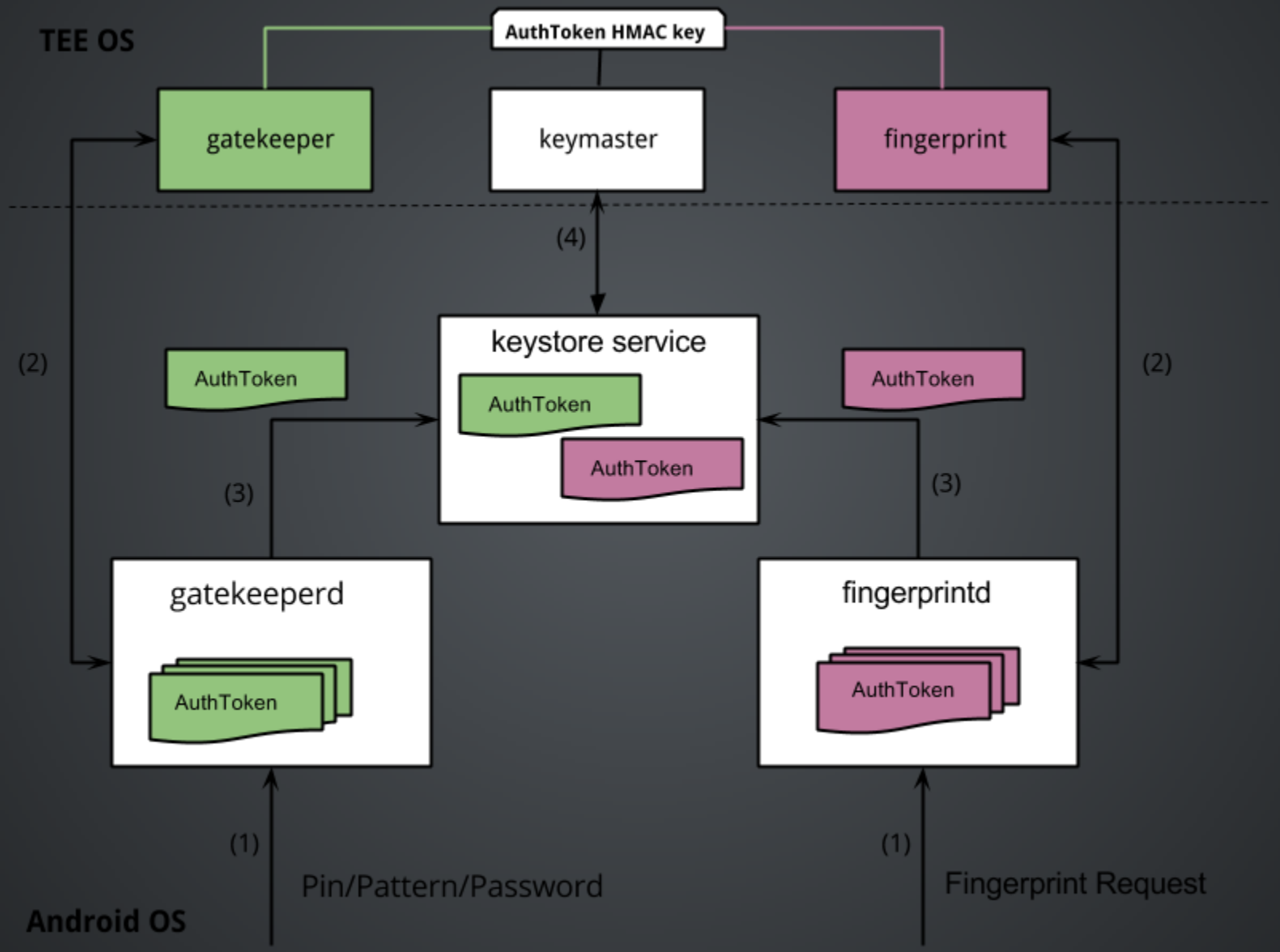
AUTHENTICATION

- Android 6.0 introduces a new Hardware Abstraction Layer (HAL) for hardware-based security
- Used by Fingerprint API, Lockscreen, Device Encryption and Client Certificates
- Protect keys against kernel compromise or physical attacks
- **Keystore:** hardware-backed storage for keys, usually including a Trusted Execution Environment (TEE)
- **Gatekeeper:** Components for PIN/pattern/password authentication
- **Fingerprint:** Components for fingerprint authentication

ARCHITECTURE

- Gatekeeper and Fingerprint components interact with the Keystore through the use of Authentication Tokens
- At first boot, a 64-bit User SID (Secure IDentifier) is created from Gatekeeper information
- SID identifies the user and is the token used to access his cryptographic material
- AuthTokens contain the SID
- Hardware enforces a minimum amount of time between authentication attempts in order to avoid bruteforcing
- Keys don't leave the TEE

AUTHENTICATION FLOW



FULL DISK ENCRYPTION

FILESYTEM ENCRYPTION

- Full FS encryption can be enabled on Android devices since Android 3.0.
- Android 5.0 improves FS encryption: faster encryption, encrypt on first boot, patterns and encryption without password, hardware-backed storage of keys
- Uses 128-bit Advanced Encryption Standard (AES) with cipher-block chaining (CBC). 256-bit key is reccomended for optimal security

SECURITY THREATS

ANDROID SECURITY THREATS

- Permission mechanism is too coarse
- Vendors don't support old devices (no security fixes)
- Privilege escalation using old kernel bugs (that's what rooting Apps do)
- APK repacking with malware
- Apps signed with the same certificate can leverage the shared UID to share sensitive data
- Bugs in the Trusted Execution Environment implementations (Qualcomm's had serious security flaws)

CASE STUDY: STAGEFRIGHT

STAGEFRIGHT

- Group of bugs in the Stagefright component of AOSP
- Discovered by Joshua Drake of the Zimperium security firm
- Announced on July 27, 2015
- Allows an attacker to perform remote code execution and privilege escalation on a device
- Affects all Android versions from Froyo (2.2) to Lollipop (5.1.1): 95% of devices at the time
- Exploits integer overflows of libstagefright's MP4 parsing code

IMPACT

- Bugs are triggered by simply playing an MP4 video
- Can be triggered automatically by an MMS thanks to Hangout's video pre-loading
- Requires no user interaction
- Totally invisible to the user, as an exploit can erase the MMS through code
- Requires an Over-The-Air (OTA) update from the phone manufacturer to get fixed
- **First fixes didn't actually fix the problem**

RUNNING PRIVILEGES

- Luckily, stagefright doesn't run as **root**
- Still, it runs inside the media server
- More privileges than normal Apps: camera, audio, sockets, bluetooth
- On some devices, even more: graphic devices, sdcard_r, internal media R/W, adb shell

CONCLUSION

CONCLUSION

- Android security is a though problem, but it's improving
- Developers must be careful, especially when using NDK
- Users should not install APKs from 3rd parties (e.g.: cracked APKs)
- Rooting Apps are basically exploiting your OS
- If a rooting App can run code as **root**, any App can
- Keep your phone updated :)

BIBLIOGRAPHY (1)

- Android Security
- P. Faruki et al., "Android Security: A Survey of Issues, Malware Penetration, and Defenses"
- Felt, Adrienne Porter, et al. "A survey of mobile malware in the wild." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011.

BIBLIOGRAPHY (2)

- Vidas, Timothy, Daniel Votipka, and Nicolas Christin. "All Your Droid Are Belong to Us: A Survey of Current Android Attacks." WOOT. 2011.
- Stagefright: Scary Code in the Heart of Android, Black Hat USA '15 (slides) (video)
- Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption

THE END