Lecture 2

PLATFORM SECURITY IN
ANDROID OS
You will be learning:

- Android as a software platform
  - Internals and surrounding ecosystem

- Security techniques in Android:
  - Application signing
  - Application isolation
  - Permission-based access control
  - Hardware-based security features
Mobile Software platform security

Platform Security Architecture

Mobile Device
- Third-party library
- Application
- Third-party service
- System service
- System library

Platform Security Component
- Reference Monitor
- IPC
- Software Isolation

HW/Security API
- Application Installer
- Application Loader
- Execution Protection

Device Management
- System Updater
- Policy Database
- Application Database

Secure Storage Provider
- Boot Verifier
- Secure Storage
- Legacy DAC

Legacy DAC
- Device Identification
- Isolated Execution
- Device Authentication

Legend
- Role
- Platform Security Component
- Third-Party Software Component
- Hardware-Security Functionality

User
- Developer
- Centralized Marketplace Operator
- Auxiliary Marketplace Operator

Platform Provider
- Administrator
Mobile software platforms

- Which mobile platforms have you heard of?
Smartphone platforms

Android: 87.6%
iOS: 11.7%
Windows Phone: 0.4%
Other: 0.3%

(IDC 2016. [Smartphone OS Market Share, 2016 Q2](https://www.idc.com/getdoc.jsp?containerId=prUS27198616))
Android in a nutshell

- Linux-based (ARM, x86, x86_64, MIPS)
- Widely used for phones and tablets
  - Wearables, smart TVs, cameras, (handheld) gaming consoles, etc.
- Open-source software stack + closed source applications and services
Security goals

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

- Linux = the kernel
- “Desktop Linux” ≈ GNU / Linux
- Linux DAC = (Unix) file permissions
- Linux MAC = SELinux
- Permissions = Android app perms.
Android Software Stack

Android Open Source Project. Security. 2015
Android Software Stack

Reference monitors

System Applications
Third-party Applications
Manager IPC Proxies
Content Providers
System Services & Managers
System Server
Media Services
Media Server
Java Core Library
Core Libraries
ART
Android Runtime
ART
Dalvik
Linux Kernel
Drivers
Binder
SELinux
Power Management

Android Open Source Project. Security. 2015
Android Software Stack

Android OS

- System Applications
- Third-party Applications
- Manager IPC Proxies
- Content Providers
- System Services & Managers
- Native Services
- Native Libraries
- HAL Modules
- Drivers
- Binder
- SELinux
- Power Management

Media Server

- Media Services

Native Code

- Native Code
- ART
- Dalvik

Core Libraries

- Java Core Library

Application Framework

- Application Framework

Applications

- Applications

Trusted Execution Environment

- Trusted Execution Environment

Trusted OS Services

- Trusted Applications

Trusted OS Kernel

- Trusted OS Kernel

Hardware

Google. Android for Work Security white paper. 2015
Application components

- Activity
  - Visible UI
- Service
  - Background job
- Content Provider
  - Database
- Intent
- Broadcast Receiver
- Manifest
- Signature

Android Open Source Project. Application Security. 2015
Software distribution

- Apps from multiple sources
  - Google Play
  - Auxiliary marketplaces
  - Sideloadung
  - Pre-installed software

- Marketplace services
  - Discovery
  - Purchase & Installation
  - User-submitted ratings / flagging
  - Malware scans (Google Bouncer)
  - Remote application installation & removal
Application signing

- Goal: same-origin policy for apps

Developer signs with private key

Signature verified with developer’s public key
Application signing (cont.)

- For application packages (APKs)
  - Self-signed X.509 certificates (no PKI!)
  - Individual signature for each file part of APK
  - Package update requires same certificate
- For over-the-air updates (OTAs)
  - Signature over entire file stored in ZIP comment
  - Verified by OS and Recovery Mode
- System images must be signed (since 6.0)

Android application packages

Com.example.app.apk

META_INF
  - AndroidManifest.XML
  - classes.dex
  - lib
  - res
  - assets

MANIFEST.MF
CERT.RSA
CERT.SF

native code
non-compiled resources
application assets
Package Installation

- Code and resources (common)
  - `/data/app/com.example.app/`
  - `lib/<arch>/libapp.so`
  - `oat/<arch>/base.odex`
  - `base.apk`

- Data (per user)
  - `/data/user/0/com.example.app/`
    - `files/`
    - `databases/`
    - `shared_prefs/`
  - `/data/user/1/com.example.app/`
    - `...`
Package management components

- Package Installer (system app)
- pm command
- PackageManager
- Installer
- MountService
- MediaContainerService
- AppDirObserver
- /dev/socket/installd
- /dev/socket/vold
- installd daemon
- vold daemon
- /system
  - framework/
  - app/
  - priv-app/
  - vendor-app/
- /data
  - packages.xml
  - packages.list
  - app/
  - data/
  - media/
  - user/

Application isolation

- Goal: Applications cannot interfere with one another
Application isolation

Implementation on Android:

- Kernel: Process & memory protection
- Kernel: Linux DAC
- Kernel: Linux MAC (SELinux)
- Middleware: mediation of Binder IPC
- Applications run in separate ART virtual machine instances
Application Sandbox

- Each application assigned a Unix UID
  - One UID per user per application (since 5.0)
  - UID owns
    - Filesystem resources in /data/user/<nr>/
    - Processes
    - Permissions (!)

- Applications from same developer (= signed with same developer key) may share UID sandbox
Application isolation

- Linux DAC domain (UID)

- System
  - Services
- Third-party applications
  - Applications
Rooting

- Rooting applications exploit vulnerabilities in privileged system daemons to obtain shell.

- Note: bootloader unlocking intentionally supported by many OEMs.
  - e.g. `fastboot oem unlock`
SELinux in Android

- Goal: System services and applications should not be able to deviate from their intended *modus operandi*
SELinux in Android (cont.)

- Implementation on Android:
  - Kernel-level MAC (SELinux) – Policies based on SELinux context
  - Middleware MAC (MMAC) – Policies based on package identity

SELinux in Android (cont.)

- Enforces MAC even for processes running with root/superuser privileges (since 4.4)
- Blocks many root exploits and misconfigurations
- Cannot protect against kernel exploits

Smalley, Craig. [Smalley, Craig: Security Enhanced (SE) Android](https://example.com). 2013
SELinux in Android

- Type Enforcement
  - Access Control Policy described as rules on abstract labels
  - System processes (subjects) and system resources (objects) mapped to labels
SELinux in Android

- *Domain* - Label for process(es)
- *Type* - Label for object(s)
- *Class* - Kind of object being accessed
  - (e.g. file, socket)
- *Permission* - Operation being performed
  - (e.g. read, write)

SELinux Project Wiki. [ObjectClassesPerms](#)
SELinux rules

- EVERYTHING FORBIDDEN BY DEFAULT!

- ALLOW rules define how subjects may interact with objects

- NEVERALLOW rules prevent specific ALLOW rules from being added to a policy
SELinux rule structure

ALLOW [domain] [type] : [class] { [ set of permissions ] }

Subject (e.g. process)
Object (resource)
Class of resource (e.g. file, socket, directory)
Operations allowed by this rule on the Class

NEVERALLOW [domain] [type] : [class] { [ set of permissions ] }
Application isolation (cont.)

• Linux DAC domain (UID)
• Linux MAC domain (SELinux)
Protected APIs

- Unrestricted service calls
- Approval-based service calls
- Restricted service calls
Protected APIs

- Goal: Protect system resources from unauthorized access
Protected APIs (cont)

- Implementation in Android:
  - Protected APIs for "risky" actions
  - Permission-based (mandatory) access control
Protected APIs (cont)

- What kinds of system calls on a smartphone would warrant protecting and why?
Examples of Protected APIs

- Changing device wallpaper, ringtone
- Making phone calls, sending SMS’s
- Using camera, microphone, GPS
- Internet, wireless, Bluetooth access
- Reading/writing contacts, SMS log
- Rebooting device
- Factory reset

Sensitive user data

- Subject to permissions checks:
  - Personal information (e.g. contacts)
  - Sensitive input devices (e.g. camera)
    - Location tracking can be manually disabled
  - Device metadata (e.g. logs,)
Access control & permissions

- Goal: Controls application access to protected APIs (and each other)
  - User agency vs. protecting system resources
  - Usability of security features
Android permissions

- 4 categories
  - Normal
  - Dangerous
  - System
  - Signature or System
Examples of Protected APIs

- Changing device wallpaper, ringtone
- Making phone calls, sending SMS’s
- Using camera, microphone, GPS
- Internet, wireless, Bluetooth access
- Reading/writing contacts, SMS log
- Rebooting device
- Factory reset

Permission assignment

- Application declare **all permissions** in `AndroidManifest.xml`
- Permissions assigned to **application UID**
- Some permissions not user-grantable
  - Only available to pre-installed applications
Permission assignment

- **Normal** permission granted automatically
- **Signature** permissions granted if app signature matches the declarer of the permission
- **System** permission only assignable by OEM
User approval (up to 5.1)

- **Dangerous** permissions require user approval at install time
- If not granted, application not installed at all!
- Granted for all users
  - Stored in `/data/system/packages.xml`

User approval (since 6.0)

- **Dangerous** permissions require user approval at runtime
- If not granted, application continues to run with limited capabilities
- Permission managed per application, per user
  - Stored in `/data/system/users/<id>/runtime-permissions.xml`
Scoped directory access (since 7.0)

- Allow access to specific shared storage directories

Permission revocation

- May be revoked later from application settings
Alternatives to obtaining permissions

- Delegate task to other application using `Intent`, e.g. invoke Camera app using `ACTION_IMAGE_CAPTURE` Intent
  - Caller does not need `CAMERA` permission
  - Caller cannot control the user experience, but does not have to provide UI for task
  - If no default app available, user is prompted to designate the handler

Android Open Source Project. Permissions Best Practices.
Intents

- Messaging object used for **Inter-Component Communication (ICC)**
  - Recall: activities, services, content providers, broadcast receivers

- Addressing
  - **Explicit** – fully qualified component name
  - **Implicit** – Intent filter declared in manifest
    - Provides a mechanism for late binding

- **Pending Intents**
  - Token-based access control delegation
Binder

- IPC system for object-orientated operating system services (comp. CORBA/COM)
- Most underlying IPC based on Binder
  - Intents & content providers abstractions on top of Binder
  - Cf. local UNIX-domain sockets, signals, filesystem
  - Bionic libc doesn’t support System V IPC
- Does not provide mediation by itself
  - Access mediated by system services
Binder Service Discovery

Binder Driver (/dev/binder)

ServiceManager (servicemanager)
- binder_become_context_manager()
- ioctl(BINDER_SET_CONTEXT_MGR)
- do_add_service()
- do_find_service()
- Services List (*svclist)

System Server (system_server)
- Package Manager
- Activity Manager
- Other System Services & Managers

Application(s)
- Application Components e.g. Activity, Service etc.
- Framework library

Other System Services & Managers

Linux Kernel

1 2 3 4
Starting an Activity

1. Application
   - Component e.g. Activity, Service etc.
   - Activity
   - startActivity(intent)

2. Binder

3. System Server (system_server)
   - Package Manager
   - Manifest
   - checkCallingPermission(...)
   - PERMISSION_GRANTED

4. Other System Services & Managers
   - Activity Manager
   - transact(...)
Starting an Activity

Application

Component e.g. Activity, Service etc.

SecurityException

startActivity(intent)

RPC Stub
Application Framework

System Server (system_server)

Package Manager

checkCallingPermission(...)

Activity Manager

Other System Services & Managers

ICK Reference Monitor

transact(…)

Binder

Linux Kernel

PERMISSION_DENIED

1

2

3

4
Authentication

- **Keyguard**
  - Pattern
  - PIN / Password

- **Gatekeeper HAL (since 6.0)**
  - Allows Keyguard to make use of native security features

- **TrustAgentAPI (since 5.0)**
  - Enables services that notify the system about whether they believe the environment of the device to be trusted

Authentication (cont.)

- Smart Lock Trust Agent (since 5.0)
  - Trusted Bluetooth device
  - Trusted NFC
  - Trusted place (via geofencing)
  - Facial recognition
  - On-body detection

- Fingerprint HAL (since 6.0)
  - Access to vendor-specific fingerprint hardware

Nexus Help: [Set up your device for automatic unlock](http://developer.android.com/training/connect-devices/detect-device.html).
Android Developers. [Creating and monitoring Geofences](http://developer.android.com/training/location.html).
Hardware-based security features

- Goals:
  - Secure storage
  - Platform integrity
Hardware-based security features

- Implementation on Android:
  - Keychain / Keystore
  - Full-disk / File-Based encryption
  - Verified boot
Keychain

- System credential store for private keys and certificate chains (since 4.0)
- KeyChain API is used for Wi-Fi and Virtual Private Network (VPN) certificates
- Hardware-backed keystore binds keys to device to make them non-exportable
  - KeyInfo.isInsideSecureHardware() (since 6.0) indicates if key is stored in hardware keystore

Google. Android for Work Security white paper. 2015
Keystore

- Keystore for application-bound keys
  - Access via Java CE API
- KeymasterHAL
  - Access to hardware-backed keystore
  - Asymmetric key generation, signing and verification (since 4.1)
  - Binder IKeyStoreInterface (since 4.3)
  - Symmetric key support, access control, public key import and private / symmetric key import (since 6.0)
- Key Attestation (since 7.0)


Android Developers. Android 7.0 for Developers.
Android Developers. Key Attestation.
Keystore (cont.)

Full Disk Encryption

- Block-device encryption based on \texttt{dm-crypt}
- Encrypted on first boot (since 5.0)
- AES 128 CBC and ESSIV:SHA256
- DEK encrypted with AES 128
  - KEK derived from user PIN / password / pattern + hardware-bound key stored in TEE (since 5.0)
- Crypto acceleration through hardware AES (e.g. \texttt{dm-req-crypt})

File-Based Encryption (since 7.0)

- Direct Boot enables supported apps to operate before user unlocks device
- Two storage locations for data:
  - Credential Encrypted (CE) storage (default) only available after user has unlocked device
  - Device Encrypted (DE) storage available during Direct Boot and after user has unlocked device (requires hardware-backed Keymaster)

**Verified Boot**

- Based on `dm-verity` kernel feature
- Calculates SHA256 hash over every 4K block of the system partition block device
  - Hash values stored in hash tree
  - Tree collapsed into a single root hash
- Hashes verified on-demand on disk access
- Signature of the root hash verified with public key included on the boot partition
  - Must be verified externally by the OEM

Verified Boot Hash Tree

Android Open Source Project. **Verified Boot**. Cryptsetup. **DMVerity**.
Mobile Software platform security

Platform Security Architecture

Mobile Device
- Third-party library
- Application
- Third-party service
- System service
- System library

Platform Provider
Administrator

System Updater
Reference Monitor
IPC
Software Isolation
Application Installer
Application Loader
HW/Security API

Device Management
Boot Verifier
Secure Storage Provider
Legacy DAC
Execution Protection

Boot Integrity
Secure Storage
Device Identification
Isolated Execution
Device Authentication

Legend
- Role
- Platform Security Component
- Third-Party Software Component
- Hardware-Security Functionality

User
Developer
Centralized Marketplace Operator
Auxiliary Marketplace Operator
Did you learn:

- Android as a software platform
  - Internals and surrounding ecosystem

- Security techniques in Android:
  - Application signing
  - Application isolation
  - Permission-based access control
  - Hardware-based security features

Contributors: Thomas Nyman, Filippo Bonazzi, Sini Ruohomaa, N. Asokan
Plan for the course

- Lecture 1: Platform security basics
- Lecture 2: Case study – Android
- Lecture 3: Mobile platform security
- Lecture 4: Hardware security enablers
- Lecture 5: Usability of platform security
- Extra lecture: IoT Security
- Invited lecture: SE Android policies
- Extra lecture: Machine learning and security
- Lecture 6: Summary and outlook