CS412 Software Security

Mobile Security

Mathias Payer

EPFL, Spring 2019

Mobile computing

You are tasked in developing a mobile computing system.

- What features do you need?
- How do you ensure device security?
- How do you protect user data?
- How do you make money?

Android statistics

- Over 2 billion active users (0)
- Android generates $31 billion revenue (2)
- 4,000+ different devices (1)
- Android OS with highest crash rate: Gingerbread (3)
- Percentage of Android devices running Marshmallow: 1.2% (1)
- Percentage of Android devices running Lollipop: 34.1% (1)

(0 Ubergizmo, 5/17/17, 1 DMR stats, 9/29/15, 2 Bloomberg, 1/21/16, 3 Greenbot, 3/27/14)

Android history

- 2005: Google buys Android
- 2007: Initial SDK released
- 2008: First devices announced
- 2009: Cupcake (1.5, 3), Donut (1.6, 4), Eclair (2.0, 5)
- 2010: Froyo (2.2, 8), Gingerbread (2.3, 9)
- 2011: Honeycomb (3.0, 11), Ice Cream Sandwich (4.0.1, 14)
- 2012: Jelly Been (4.1.1, 16)
- 2013: KitKat (4.4, 19)
- 2014: Lollipop (5.0, 21)
- 2015: Marshmallow (6.0, 23)
- 2016: Nougat (7.0, 24-25)
- 2017: Oreo (8.0 - 8.1, 26-27)
- 2018: Pie (9.0, 28)
- 2019: Android Q? (10.0, 29)

Android security goals

- Isolate individual applications
- Protect system resources from applications
- Vet applications "online"
- Protect data of *the* user (until 5.0 single user)

Android security architecture

- Applications are carefully vetted server-side and only approved applications can be installed from the "market"
- Each application runs in a Java-like sandbox, restricted to user-granted permissions
- Applications leverage well-defined API channels to communicate with other applications
- The system is hardened against local user (app-based) attacks.

Android app security

- Each apps runs in its own secure context/sandbox
- Interactions between apps are restricted through the API
- Each app has an associated policy, encoding the permissions
- Apps are signed by the developer, vetted, and installed from a central market

Android system security (1/3)

Android leverages the Linux kernel as a foundation. Such an established kernel brings the following advantages:

- Process isolation
- User-based permission model (with RBAC extensions)
- Extensible mechanism for secure IPC
- Configurable (trim down kernel to limit exposure)
- Effective resource isolation when accessing hardware

Bonus: hardware drivers readily available, well-established development environment

Android system security (2/3)

Isolation:

- Each application runs as its *unique user*
- Stringent permissions on file systems
- Hardened Linux kernel protects applications
- SELinux to apply access control policies on processes

Android system security (3/3)

Trust and safe defaults:

- Verified boot and file system encryption
- The root partition with code and system configuration is mounted read-only
- Crypto API provides secure implementations and key management
- User-space: stack canaries, integer overflow mitigation, double free protection (through allocator), fortify source, NX, `mmap_min_addr`, ASLR, PIE, relro, immediate binding
- Each release: security updates, patches, toolchain updates, tighter security defaults

Android permissions

Android leverages a complex permission system on a per-app basis. Android splits permissions into several categories, based on the potential risk to the customer

- PROTECTION_NORMAL: no risk, allowed by default (if declared)
- PROTECTION_SIGNATURE: permission granted if app is signed with the same key as app providing the service (this type of permission is for cross-app communication)
- PROTECTION_DANGEROUS: this permission has potential implications to the user's privacy, ask for explicit permission through a pop-up

Android default permissions (1/2)

ACCESS_LOCATION_EXTRA_COMMANDS
ACCESS_NETWORK_STATE ACCESS_NOTIFICATION_POLICY
ACCESS_WIFI_STATE **BLUETOOTH BLUETOOTH_ADMIN**
(pairing without user interaction is privileged though)
BROADCAST_STICKY CHANGE_NETWORK_STATE
CHANGE_WIFI_MULTICAST_STATE CHANGE_WIFI_STATE
DISABLE_KEYGUARD EXPAND_STATUS_BAR
FOREGROUND_SERVICE GET_PACKAGE_SIZE
INSTALL_SHORTCUT INTERNET
KILL_BACKGROUND_PROCESSES MANAGE_OWN_CALLS
MODIFY_AUDIO_SETTINGS

Android default permissions (2/2)

NFC READ_SYNC_SETTINGS READ_SYNC_STATS
RECEIVE_BOOT_COMPLETED REORDER_TASKS
REQUEST_COMPANION_RUN_IN_BACKGROUND
REQUEST_COMPANION_USE_DATA_IN_BACKGROUND
REQUEST_DELETE_PACKAGES
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS SET_ALARM
SET_WALLPAPER SET_WALLPAPER_HINTS **TRANSMIT_IR**
USE_FINGERPRINT VIBRATE WAKE_LOCK
WRITE_SYNC_SETTINGS

Android dangerous permissions

- *Calendar:* READ_CALENDAR WRITE_CALENDAR
- *Call log:* READ_CALL_LOG WRITE_CALL_LOG PROCESS_OUTGOING_CALLS
- *Camera:* CAMERA
- *Contacts:* READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
- *Location:* ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
- *Microphone:* RECORD_AUDIO
- *Phone:* READ_PHONE_STATE READ_PHONE_NUMBERS CALL_PHONE ANSWER_PHONE_CALLS ADD_VOICEMAIL USE_SIP
- *Sensors:* BODY_SENSORS
- *SMS:* SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
- *Storage:* READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Android missing permissions

Accessing sensors (temperature, accelerometer, gyroscope, . . .) do not require permissions.

Android missing permissions

Accessing sensors (temperature, accelerometer, gyroscope, ...) do not require permissions.

Security solution: disable sensors if the app is not in the foreground.

Android intents

Google's idea of mobile IPC

> *An Intent is a simple message object that represents an "intention" to do something. For example, if your application wants to display a web page, it expresses its "Intent" to view the URL by creating an Intent instance and handing it off to the system. The system locates some other piece of code (in this case, the Browser) that knows how to handle that Intent, and runs it. Intents can also be used to broadcast interesting events (such as a notification) system-wide. (From Google's Android website.)*

Android attack vectors: intents

- Unauthorized intent receipt: attacker creates an intent filter, receives other apps' intents that contain privileged information (e.g., intent filter for web service intercepts online payment process)
- Intent spoofing: attacker sends a malicious intent to an intent processor (e.g., flooding the network with malicious messages)

Android attack vectors: intents

- Unauthorized intent receipt: attacker creates an intent filter, receives other apps' intents that contain privileged information (e.g., intent filter for web service intercepts online payment process)
- Intent spoofing: attacker sends a malicious intent to an intent processor (e.g., flooding the network with malicious messages)

- Ongoing work to defend against such attacks, e.g., by restricting communication among signed or trusted components.

Android attack vectors: communication

- Insecure internet communication: run Wireshark to intercept traffic
- Bluetooth, NFC, IR is not restricted: peripherals are not protected against malicious apps, any app may access any paired device. This is an issue of the granularity of the privileges. There's only one Bluetooth privilege, privileges are per *app*, not *per connected device*.

Android attack vectors: privileges

- Overprivileged app: confused deputy, bugs in application can be leveraged by attacker to gain privileges

Historic Android attack vectors

- Insecure storage: there were no access restrictions on the SD card (why?), an attacker may read/write any data on the SD card (fixed in Android 5.0)

Android's central security strategy

The central market is a core security strategy

- Developers must sign apps, making them identifiable, increasing switching cost
- Apps can be vetted based on developer profiles
- Apps are verified using static and dynamic checks before making them public
- Telemetry allows Google to tie malicious apps to companies/developers

Summary and conclusion

- Android security evolved over time
- Android systems hardened against exploits
- Developers sign apps which identifies them (comes with a cost)
- Applications are vetted centrally and installed from the market