INF226 – Software Security

Håkon Robbestad Gylterud

2019–11–06

Mobile security

Phones

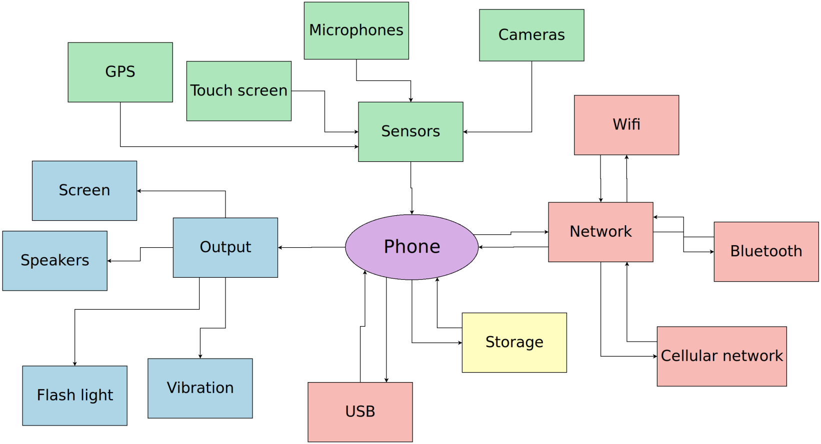

Figure 1: Mobile phones

Special concerns

Remote attack vectors:

- SMS
- Telephone
- Base stations
- WiFi

Mobile applications

Is downloading an app different from visiting a web-page?

Mobile applications

- Can access:
  - Sensors
  - Network
  - Storage
- Runs bytecode as OS process
- Can communicate with other processes

Threat

Mobile phones are increasingly valuable targets:

- Store a lot of personal information, which can be used for
  - **advertisement**
  - **phishing** (Example: IPhone contact permission)
  - **extorision**
- Store a lot of organisational data:
  - Contacts
  - Calendar
  - Documents (**trade secrets**)
- Used for browsing web-sites:
  - Session cookies stored on phone.

- Sensors can be used for **surveillance**.
- Computation power:
  - **Coin-mining malware**.
- Connected to billing systems (NFC, phone bill)
- Two-factor authentication tokens.

Threat model of mobile applications

What must we assume of an attacker?

- Craft SMS messages?
- Control the network?
- Run their own app on the phone? (malware)
- Access the phone physically?

Mobile network security

Mobile networks are only encrypted from phones to the base station.

- Encryption based on the A5 family of block ciphers, which are known to be weak.
- Encryption can be turned off by base station through `Cipher Mode Settings`.

This means that a rogue base station (such as StingRay devices), can MITM mobile signals.

Applications must therefore use TLS or another application layer encryption to ensure transport security.

Android security

Android

Android is Google's mobile operating system:

- Based on Linux:
  - Each app has its own UID.
  - Each app has its own Linux process.
- Java as application platform
  - Each app has its own VM.

However, apps signed by the same certificate can share UID, OS process and VM.

Application components

In stead of processes, apps on Android are centered around **components**, of which there are four kinds:

- Activities
- Services
- Content providers
- Broadcast receivers

Each component is implemented by a class in Java.

Intents

Components communicate with each other through an Inter Component Commuication (**ICC**) system.

- The ICC packets on Android are called **intents**.

Intents consist of:

- **Action** string
- **Data** to operate on (URI)

There is also a basic distinction between:

- **Explicit** intents directed to a specific component
- **Implicit** intents, where the system/user finds a suitable component

Example: `ACTION_VIEW` `https://uib.no/` is an implicit intent to open a web browser on URL `uib.no`.

Intent permissions

Intent listeners (such as activities, services and broadcast receivers), are either **exported** or **private** to the application

- Exported intent listeners are accessible from any app.
- Private intent listeners are only accessible from components within the app.

This provides a quite crude access control on intents.

Activities

Activities are user interface components displayed when the user interacts with the application.

- Activities spawn other activities through intents `Activity.startActivity(Intent)`
- Applications can start activities from other applications.

Activities receive intents, and in response interacts with the user.

Example: The activity ContactEditor receives intent `EDIT content://contact/1`.

CVE-2017-7759

Firefox on Android (until v54.0) allowed access to local files via intents on remote sites:

```
<script>
location="intent:file:///(path)#Intent;type=text/html;end";
</script>
```

This allows exfiltration of local files to malicious web-sites.

Universal Cross-site scripting (UXSS)

Both Chrome (CVE-2015-1275) and Firefox (CVE-2015-7191) on Android have been vulnerable to **Universal Cross-site scripting through intents**.

Firefox example:

```
<iframe src="https://www.nsa.gov"
        onload="this.src='intent://any#Intent;
                          scheme=httpz;
                          S.browser_fallback_url
        =javascript%3Aalert%28document.domain%29;end'">
```

This allowed any site to inject scripts into any other site.

Services

Services are application components run without user interaction.

- Receives intents, just like activities.

Examples Playing music or keeping a long-lived TCP connection.

Broadcast receivers

Broadcast receivers respond to system-wide intents.

- Can be called when the application is not running.
- No GUI, except status bar notifications.
- Can launch other components.

Example: For an app to receive SMS messages it can have a broadcast-receiver for `DATA_SMS_RECEIVED`-intents.

Content providers

Content providers resolve URI's and extract data for activities and services.

How the data is stored is implementation specific:

- SQLite database
- Online storage
- Files
- ...

Content providers provide a query interface.

SQL injection in content providers

Content providers often store data in SQLite databases, but the interface provides no prevention of SQL injection attacks:

- CVE-2014-8507 : SQL injection in the queryLastApp allows remote execution of arbitrary SQL commands, and launch an activity or service.
- CVE-2018-14066: SQL injection in com.android.provider.telephony allows **access to SMS messages by unauthorised apps** (Limited to some specific phones)
- CVE-2018-9493: SQL injection in the download manager (including v9.0) content provider.

Sandboxing and encryption

Sandbox

Android processes are separated using usual Linux mechanisms:

- SELinux provides **Mandatory Access Control** (from v5.0):
  - Each application runs in its own SELinux sandbox (from v9.0)
- `seccomp` is used to **filter system calls** (from v8.0)

Encrypted storage

Android (since v5.0) uses `dm-crypt` to encrypt the phone storage.

- `dm-crypt` is a whole disk encryption utility.

Provides **confidentiality**, but **not integrity**.

Encrypted storage

Android provides encrypted storage of encryption keys through Android KeyStore:

- Either hardware based (through a **Trusted Execution Environment**):
  - Relies on phone specific ARM TrustZone (only available on some devices).
- or software based.

Malware

Threats from malware on Android

Question: What threats can we imagine malware on Android can muster?

App permissions

The App permission system regulates app-access to various resources (contacts, files, Internet, SMS, etc):

- Access control list (ACL) based
- Some permissions granted on installation
- Other permissions can be requested later
- Permissions are grouped (Example: `READ_CONTACTS` and `WRITE_CONTACTS` are grouped)

Dangerous permissions

From the Android developer documentation:

> *If the app doesn't currently have any permissions in the permission group, the system shows the permission request dialog to the user describing the permission group that the app wants access to. The dialog doesn't describe the specifi permission within that group.*

Example

App requests `READ_CONTACTS` permission.

- UI diplays: "Give The App access to the device contacts?"
- If user agrees: app is given `READ_CONTACTS` permission.

Dangerous permissions

From the Android developer documentation:

> *If the app has already been granted another dangerous permission in the same permission group, the system immediately grants the permission without any interaction with the user.*

Example

- The app has READ_CONTACTS permission
- The app requests WRITE_CONTACTS

Result: The system immediately grants that permission without showing the permissions dialog to the user.

Preinstalled software

Pre-installed software on a device can effectively bypass the need for user-granted permissions.

Threats posed by pre-installed apps:

- set up back doors
- exfiltrate personal information
- install TLS root certificates

Preinstalled software

A recent study[0] found:

- Supply chain for Android devices lacks transparency
- Found 3,118 different pre-installed apps spead over available devices.
- In some cases the pre-installed apps contained known malware

Nearly all apps with access to personal information disseminate it to third-party servers.

Pre-installed Android Software*.

Collusion between apps

The ICC system (intents) allow apps to communicate.

- There are no restrictions on which apps can commuincate

Collusion: Apps with different permissions can co-operate to attack the user.

Collision between apps

The ICC system (intents) allow apps to communicate.

- There are no restrictions on which apps can communicate

Collusion: Apps with different permissions can co-operate to attack the user.

Example

- App `A` has access to user contacts
- App `B` has access to the Internet

Now `A` and `B` can collude to exfiltrate the user contacts over the internet.

Collusion

Collusion has been detected in the wild:

- Blacso, Chen, Muttik, Roggenbach: *Detection of app collusion potential using logic programming*:
  - analysed 50,000 Android apps
  - using static binary analysis.
- They found several cases app collusion.

Libraries

In the cases where collusion has been detected, the code responsible has come from third-party libraries:

- Most likely the app developers do not know what their apps are doing!

Muddiest point

Answer on `mitt.uib.no`

Where do the Android security bugs lie?

The central tension in mobile app security:

- Sandboxing vs. inter-app communication

The intent system is quite open for *confused deputy* problems.

Where do the Android security bugs lie?

The central tension in mobile app security:

- Sandboxing vs. inter-app communication

The intent system is quite open for *confused deputy* problems.

These come in addition to the usual suspects:

- Bad access control
- XSS
- SQL injection
- Insecure deserialisation
- ...

How to secure Android applications

- Understand the intent system:
  - Implicit vs explicit
- Use the access control mechanisms to limit which components can send intents to critical services/activities.
- When sending intents across apps:
  - Consider using authentication tokens as part of URI data, to verify senders access to the data.
- Be careful with using third party libraries!