Android Application Components

When you're hunting bugs in Android apps, you're not just looking at code—you're entering a house with **different rooms, hidden doors, secret passageways, and sometimes an unlocked back gate.** That house is built out of **Android Application Components**.

ANDROID APPLICATION COMPONENTS (DEEP DIVE)

There are four main components (+ some supporting elements that make up every Android application.

1 Activities – The User's Window

An Activity is a single. focused thing the user can do. It represents a UI screen with which users interact.

Lifecycle:

- onCreate() → initialize
- onStart() → visible but not interactive
- onResume() → in the foreground
- onPause() → partially hidden
- onStop() → not visible
- onDestroy() → killed

Pentest pentest:

- Improper Data Handling
- Deep Link Hijacking
- Broken Access Control (iDOR)
- Check if Activities are **exported** in the manifest (android exported– true*) without permission checks
- Exported Activities can allow attackers to bypess login screens or trigger privileged actions without suthentication

2 Services – The Silent Workers

A Service is a background component that runs without UI:

Powes of services:

- Foreground Service → visible to user
- Background Service → runs silently
- Bound Service → allows components to bind and interact with It.

Security pentest:

- If exported, attacker can start or bind to them
- Could be used to abuse privileged actions
- Common flaws:
  - No permission checks
  - IPC injection
  - DoS

3 Broadcast Receivers – The App's Ears

A Broadcast Receiver is like a mailoyx for your app—it listens for system-wide or app-specific broadcasts.

Examples:

- Battery low varning
- Incoming SMS notification
- Custom app events

4 Content Providers – The Data Gatokeepers

Content Providers are the gatekeepers of an app's data. They control how the app's internol data is accessed by itself and by other applications.

Examples:

- Contacts provicler
- Media store
- Custom providers for app databases

Bug bounty/pentest checkljet

- Check for exported Activities/Services Receivers with no permission
- Try intent injection on exporte components
- Query Content Providers for sensitive data

If you know how each component works (and where it can break), you've already got a **map of attack surfaces**.

There are **four main components** (+ some supporting elements) that make up every Android application:

1. Activities -The User's Window

What it is:
An **Activity** is a single, focused thing the user can do. It represents a **UI screen** with which users interact. Activities are the user-facing parts of the app, and they are your first point of reconnaissance. You're not just looking for a pretty UI; you're looking for how it handles data and what it exposes. For example:

- LoginActivity → login screen
- MainActivity → main dashboard
- SettingsActivity → settings page

→ **Lifecycle:**
Activities go through states:

- onCreate() → initialize
- onStart() → visible but not interactive
- onResume() → in the foreground
- onPause() → partially hidden
- onStop() → not visible
- onDestroy() → killed

→ **Security/Pentest Notes:**

-- **Improper Data Handling**: When an activity is started, data can be passed to it via "Intents." An improperly configured activity might not validate this incoming data, leading to vulnerabilities like Cross-Site Scripting (XSS) or SQL Injection if the data is reflected or stored in a database.

-- **Deep Link Hijacking**: Deep links allow external apps or websites to open a specific activity within your target app. If not properly secured, an attacker can create a malicious link to trigger a vulnerable activity and steal sensitive information or perform actions on the user's behalf.

-- **Broken Access Control (IDOR):** An activity might be designed to display a user's profile based on a user ID in the URL. If the app doesn't properly check if the current user has

permission to view that profile, an attacker can simply change the ID to view another user's private data.

-- Check if Activities are **exported** in the manifest (android:exported="true") without permission checks.

-- Exported Activities can allow attackers to:

1. **Bypass login screens** (launch MainActivity directly).
2. Trigger **privileged actions** without authentication.

Red flag in Manifest:

```
<activity android:name=".MainActivity" android:exported="true"/>
```

- Try launching it directly.

2. Services -The Silent Workers

What it is:
A **Service** is a background component that runs without UI. Services are the app's workhorses, performing tasks in the background. Because they don't have a UI, they are often overlooked by novice bug hunters. This is where you can find some of the most critical vulnerabilities.
Examples:

- Media player running in background
- Sync service (emails, messages)
- Download manager
- A service runs in the background without a user interface. Useful for long-running or periodic tasks, it can execute even when the app's main UI is not visible.

Type	Description
Started Service	Launched via `startService()`. Runs independently, stops itself or by command.
Bound Service	Connected with `bindService()` – other components interact and receive feedback.

Type	Description
Foreground Service	Shows a persistent notification while running; used for tasks like music playback.

Types of Services:

- **Foreground Service** → visible to user (e.g., music player notification)
- **Background Service** → runs silently, but Android now heavily restricts these
- **Bound Service** → allows components to bind and interact with it

→ Security/Pentest Notes:

- If exported, attackers can start or bind to them (android:exported="true").
- Could be used to abuse privileged actions (e.g., triggering SMS send, file operations).
- Common flaws: **No permission checks**, **IPC injection**, **DoS by flooding service**.
- **Insecure Services:** A service can be configured to be "exported," meaning other applications on the device can start it and interact with it. If a service handles sensitive data or performs privileged actions without proper permission checks, a malicious app could trigger it to, for example, access private files, send a text message, or make a network request.
- **Denial of Service (DoS):** An attacker might repeatedly trigger a vulnerable service that consumes a lot of resources, causing the app to crash or the device to slow down, leading to a denial of service.

3. Broadcast Receivers - The App's Ears

What it is:

A **Broadcast Receiver** is like a mailbox for your app—it listens for system-wide or app-specific broadcasts. Broadcast Receivers are like a pager for your app. They're constantly listening for specific events, and when they hear one, they react. The problem is, sometimes they listen for events that an attacker can create.

Examples:

- Battery low warning (android.intent.action.BATTERY_LOW)
- Incoming SMS notification
- Custom app events (e.g., "com.myapp.USER_LOGGED_IN")

→ Security/Pentest Notes:

- If exported, anyone can send fake broadcasts.
- Attackers can exploit this to:
    - **Bypass logic** (e.g., send fake "payment successful" intent).
    - Cause **privilege escalation** (broadcast system-level intents).
- Always use **permissions** (android:permission) to protect sensitive receivers.
- **Insecure Broadcast Receivers:** Similar to services, a broadcast receiver can be "exported" and listen for custom broadcasts from other apps. If an attacker can craft a malicious broadcast and send it to your target app, they can trigger a hidden action. For example, a receiver might be configured to process a URL from a broadcast, leading to **Server-Side Request Forgery (SSRF)** or other injection attacks.
- **Intent Spoofing:** An attacker can send a spoofed intent to a vulnerable receiver, tricking it into performing an unauthorized action, such as logging the user out or deleting a file.

4. Content Providers - The Data Gatekeepers

What it is:

Content Providers are the gatekeepers of an app's data. They control how the app's internal data is accessed by itself and by other applications. If you can get past this gatekeeper, you can gain access to a treasure trove of information. A **Content Provider** manages app data and allows it to be shared with other apps.
Examples:

- Contacts provider (content://contacts)
- Media store (content://media)
- Custom providers for app databases (SQLite)

→ **Security/Pentest Notes:**

-- **SQL Injection:** Content providers often use an underlying database, and if they don't properly sanitize user input, an attacker can use a crafted query to bypass security and access or modify the entire database.eg (content://provider/items?filter=' OR '1'='1)

-- **Directory Traversal:** A content provider might expose files based on a user-provided path. An attacker could use ../ to traverse outside the intended directory and access sensitive files like config.xml or other application data.eg (content://provider/../../etc/passwd)

-- **Information Disclosure:** An insecure content provider can leak sensitive information, such as passwords, user data, or API keys, to other apps on the device.

-- **Check for**:

    - android:exported="true" in manifest

- Lack of permission enforcement (android:readPermission, android:writePermission)
- SQL queries built using untrusted input

Payload idea:

```
content://provider/items?filter=' OR '1'='1
```

- Boom—you might dump an entire database.

#Supporting Components

1. Intents

- Messaging system used to communicate between components.
- Two types:
  - **Explicit Intent** → directly specifies target component
  - **Implicit Intent** → system decides best handler
- Attack surface:
  - **Intent spoofing** (malicious app sends fake intent to exported component)
  - **Intent sniffing** (reading broadcast intents if not protected)

Intents allow you to:

- **Start Activities (startActivity())**
- **Start or bind to Services (startService(), bindService())**
- **Send messages (sendBroadcast())**
- **Query Content Providers (ContentResolver.query())**

2. App Manifest (AndroidManifest.xml)

- Defines which components exist.
- Declares:
  - Exported status (android:exported)
  - Required permissions (android:permission)
  - Intent filters (which intents it listens to)
- Misconfigurations here = easy privilege escalation for attackers.

#Bug Bounty/Pentest Checklist

→ Check for **exported Activities/Services/Receivers** with no permission.
→ Try **intent injection** on exported components.
→ Query Content Providers for sensitive data.
→ Look for **hardcoded secrets** in app storage (SharedPreferences, SQLite).
→ Abuse **debuggable apps** (android:debuggable="true").

 # In short:

- **Activities** → user screens (attack via exported bypasses)
- **Services** → background jobs (attack via unprotected start/bind)
- **Broadcast Receivers** → system events (attack via spoofed broadcasts)
- **Content Providers** → data storage & sharing (attack via SQLi, leaks)

Component	Function	Lifecycle	Communication
Activity	UI screen, user interaction	Six major callbacks	Intents
Service	Background work, no UI	Service lifecycle	Intents, binding
Broadcast Receiver	Listen/respond to system and app events	Minimal, `onReceive()`	Broadcast Intents
Content Provider	Secure data sharing/providers	Starts with process	Content Resolver, URIs

By - bithowl