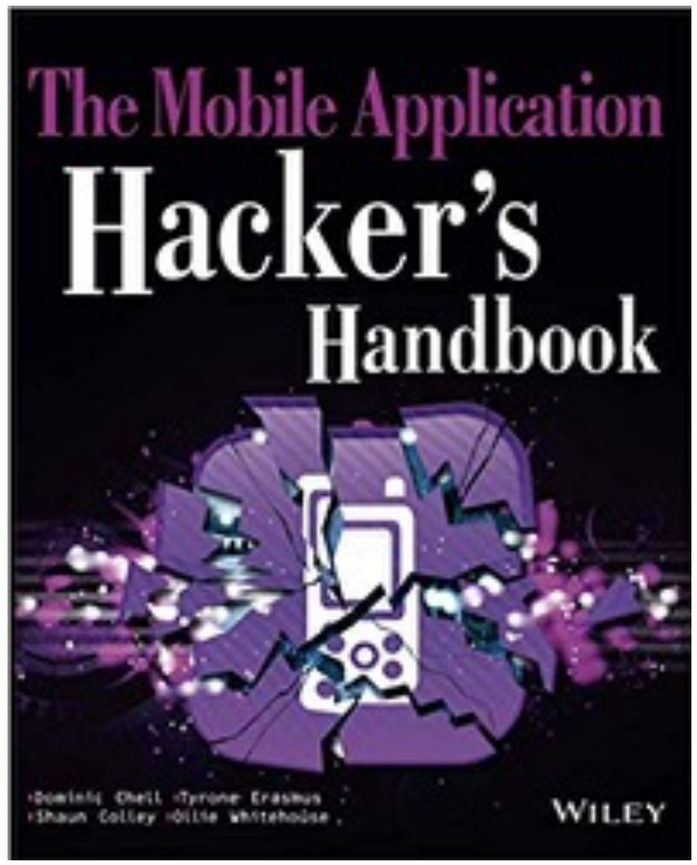
CNIT 128
Hacking Mobile Devices



7. Attacking Android Applications Part 1

Updated 9-26-22

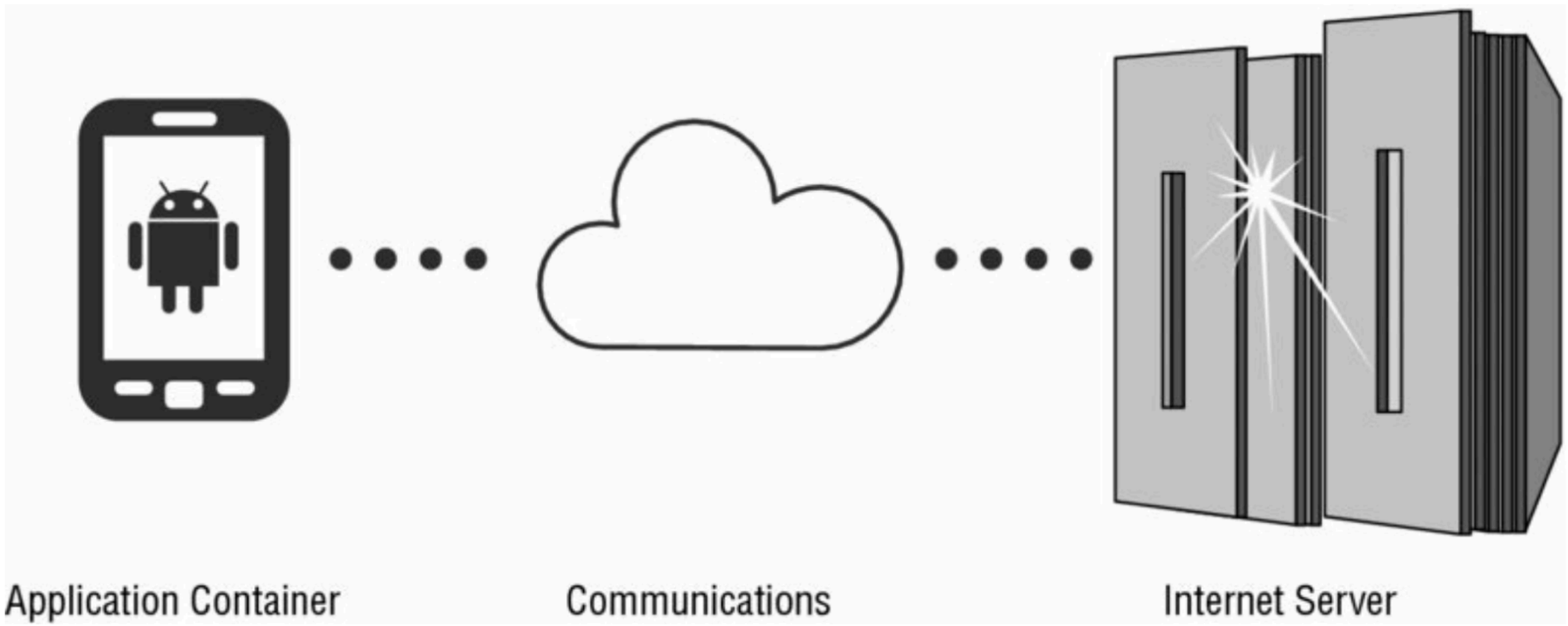
Topics

- Part 1
  - Exposing Security Model Quirks
  - Attacking Application Components
    (to p. 271)
- Part 2
  - Attacking Application Components (finishes)

Topics

- Part 3
- Accessing Storage and Logging
- Misusing Insecure Communications
- Exploiting Other Vectors
- Additional Testing Techniques

Three Main Components



Application Container

- Ways to defeat application sandbox
  - Gain access to app data
- Malicious app on a device
- Physical access to device
- Other vulnerabilities in the app

Communications

- Ways to intercept communications
  - ARP poisoning
  - Hosting a malicious wireless network
  - Compromising upstream providers

Internet Server

- Server may have vulnerabilities
- Compromised server exposes all information flowing to and from mobile apps

Exposing Security Model Quirks

Interacting with App Components

Table 7.1 The Default Export Behavior of Each Application Component Across API Versions

APPLICATION COMPONENT	EXPORTED (API < 17)	EXPORTED (API >= 17)
Activity	False	False
Broadcast receiver	False	False
Service	False	False
Content provider	True	False

targetSdkVersion

- Determines default publishing of components
- Other values: compileSdkVersion and minSdkVersion (link Ch 7a)

Android Versions in Use

Android OS market share

Android OS version	Market share
11 (Android 11)	29.9 %
10 (Android 10)	20.2 %
9.0 (Pie)	11.4 %
8.0-8.1 (Oreo)	8.2 %
7.0-7.1 (Nougat)	4.9 %
6.0 (Marshmallow)	2.3 %
5.0-5.1 (Lollipop)	2.0 %
4.4 (KitKat)	0.6 %
4.1-4.3 (Jelly Bean)	0.2 %
4.0 (Ice Cream Sandwich)	0.0 %

- Link Ch 6c, from Sept 2022

Explicitly Exported Components

- Explicitly exported

```
<receiver

    android:name="com.mahh.receiver"

    android:exported="true" >

</receiver>
```

- Unspecified; will be exported implicitly if targetSdkVersion < 17

```
<provider android:name="com.mahh.app"

          android:authorities="com.mahh.content" />
```

Implicitly Exported

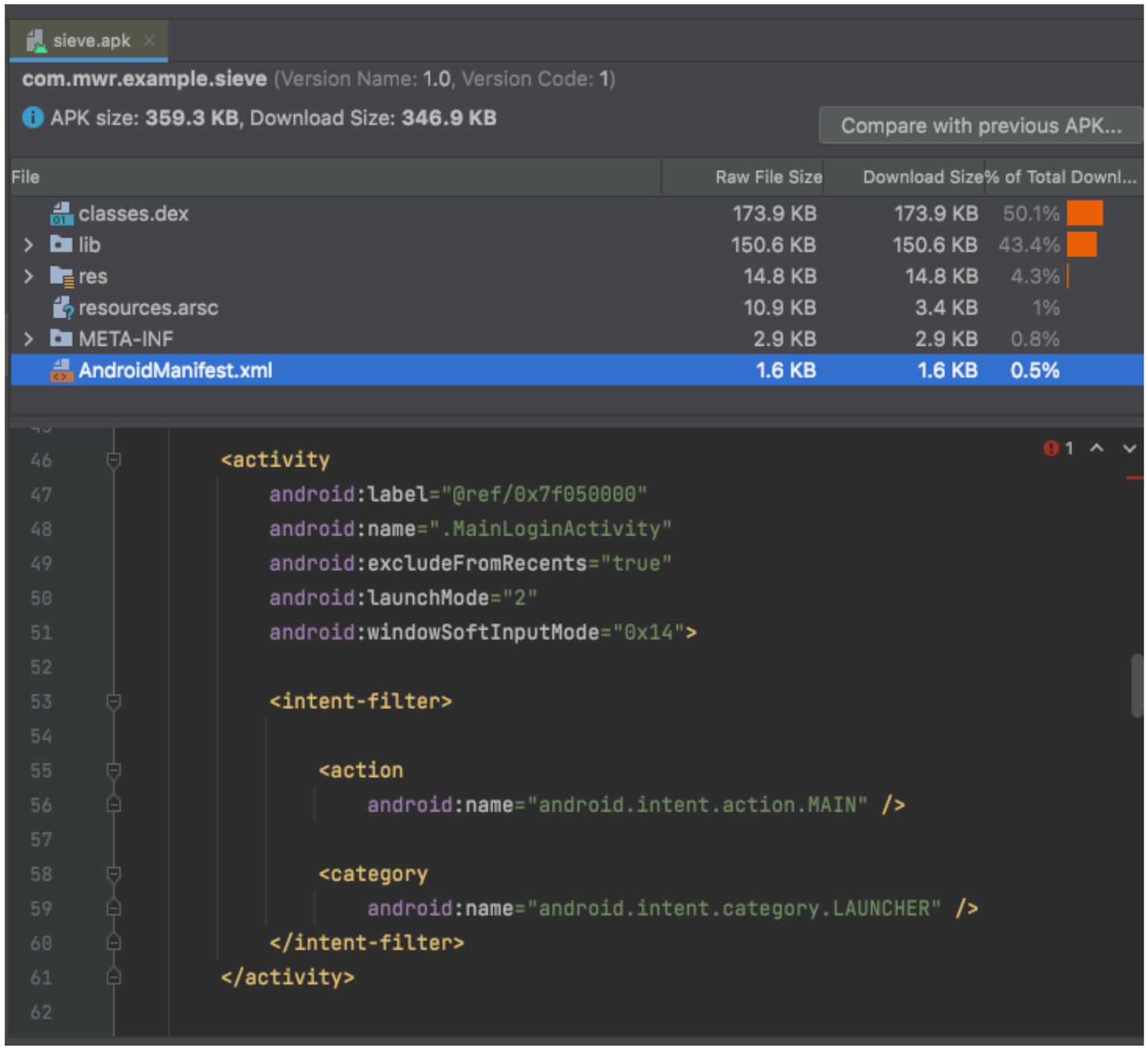
- Any component using an **<intent-filter>** is exported by default
- Like this activity

```
<activity android:name="ImageActivity">

    <intent-filter>

        <action android:name="android.intent.action.SEND"/>

        <category android:name="android.intent.category.DEFAULT"/>

        <data android:mimeType="image/*"/>

    </intent-filter>

</activity>
```

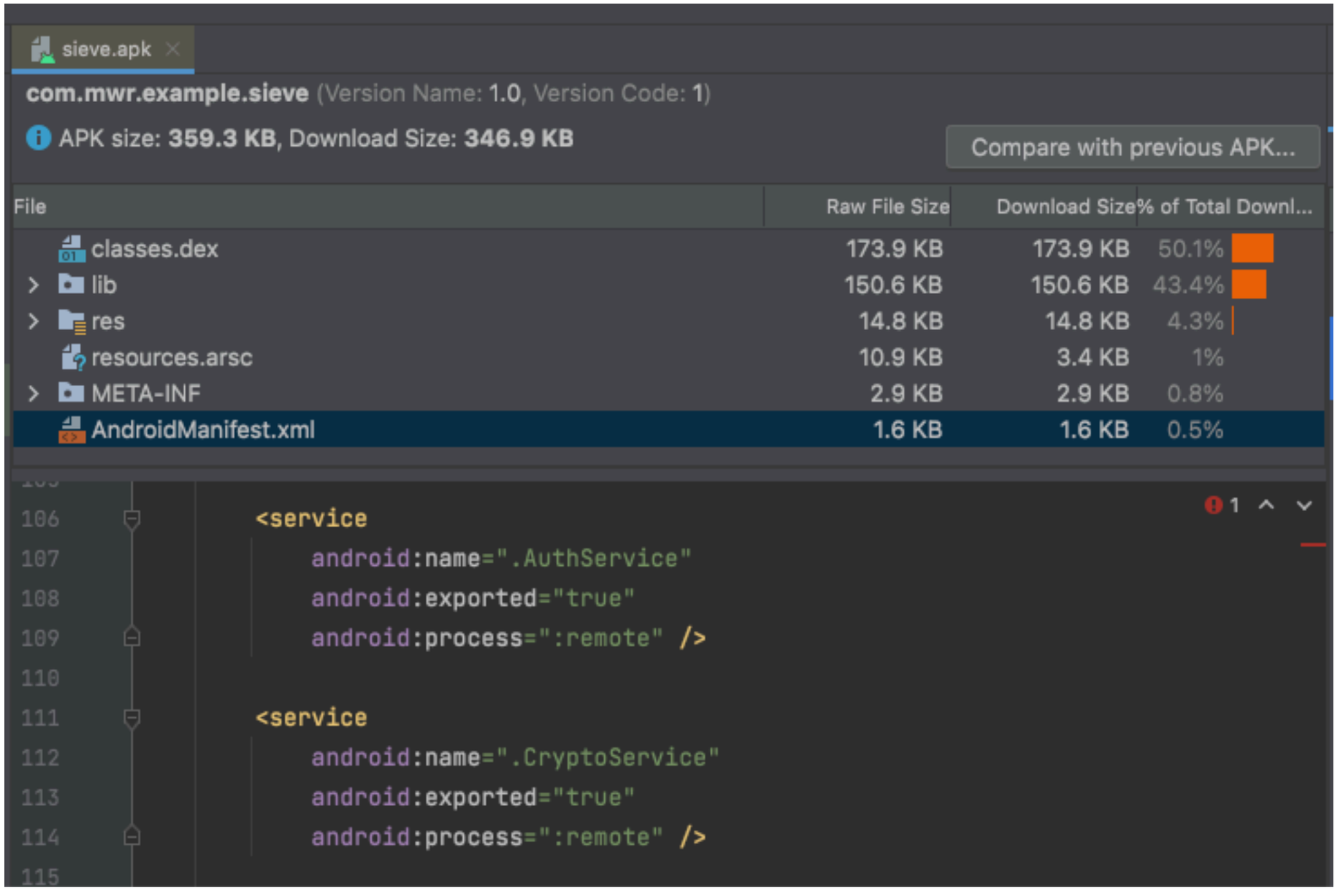
Finding Exported Components

- Examine Manifest
- https://github.com/mwrlabs/drozer/releases/download/2.3.4/sieve.apk
- Drag APK into center pane of Android Studio

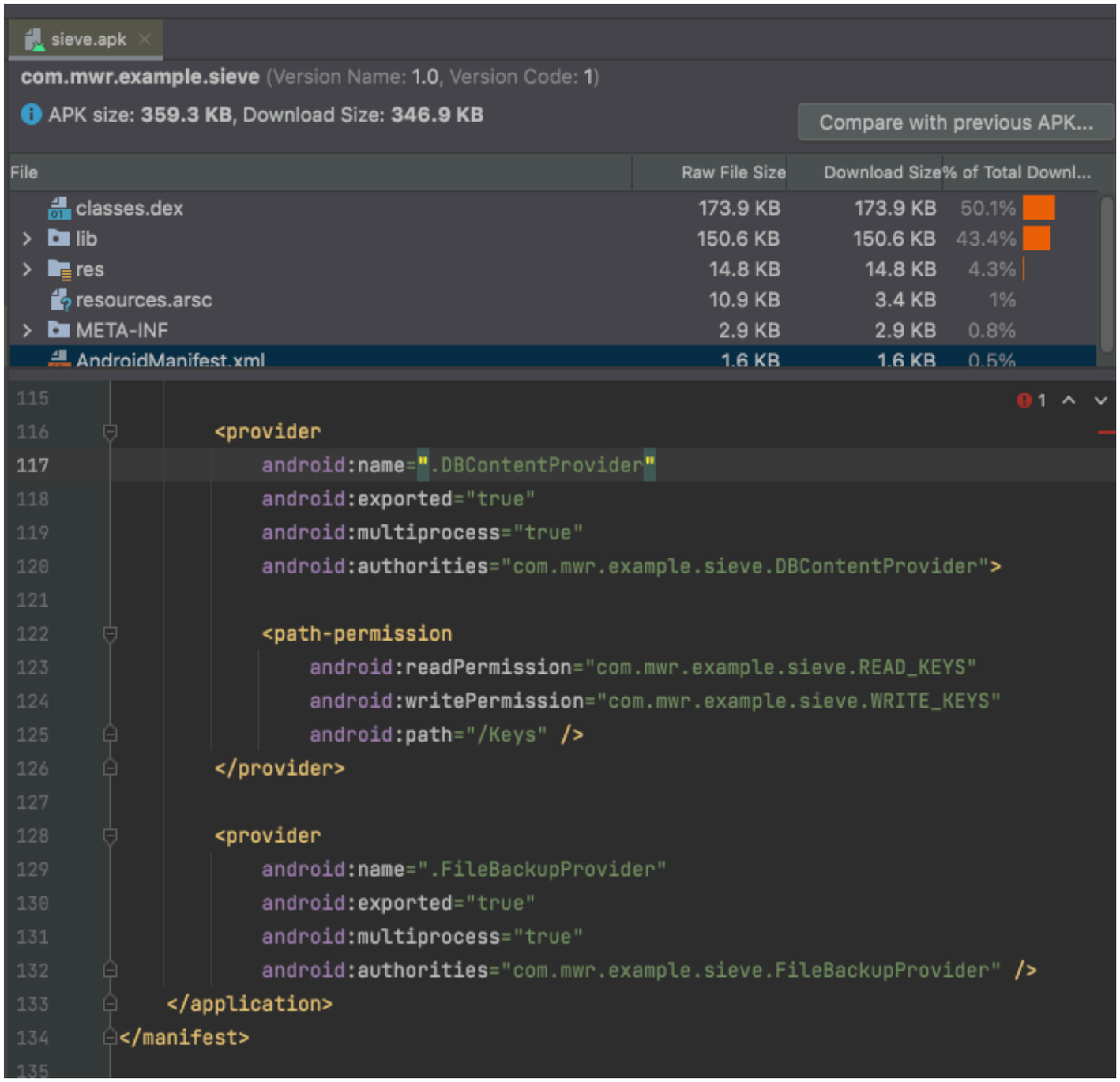
An Activity in Sieve



Services in Sieve



Content Providers in Sieve



Supreme User Contexts

- **root** and **system** users can interact with application components
  - Even when they are not exported
- Components that are not exported in the manifest are private
  - Limited to internal use by the app
  - Only attackers with root privileges can attack them

Permission Protection Levels

- Best protection is a custom permission with protection level **signature**
  - Only apps with the same signature can have that permission

Protection Level Downgrade Attack

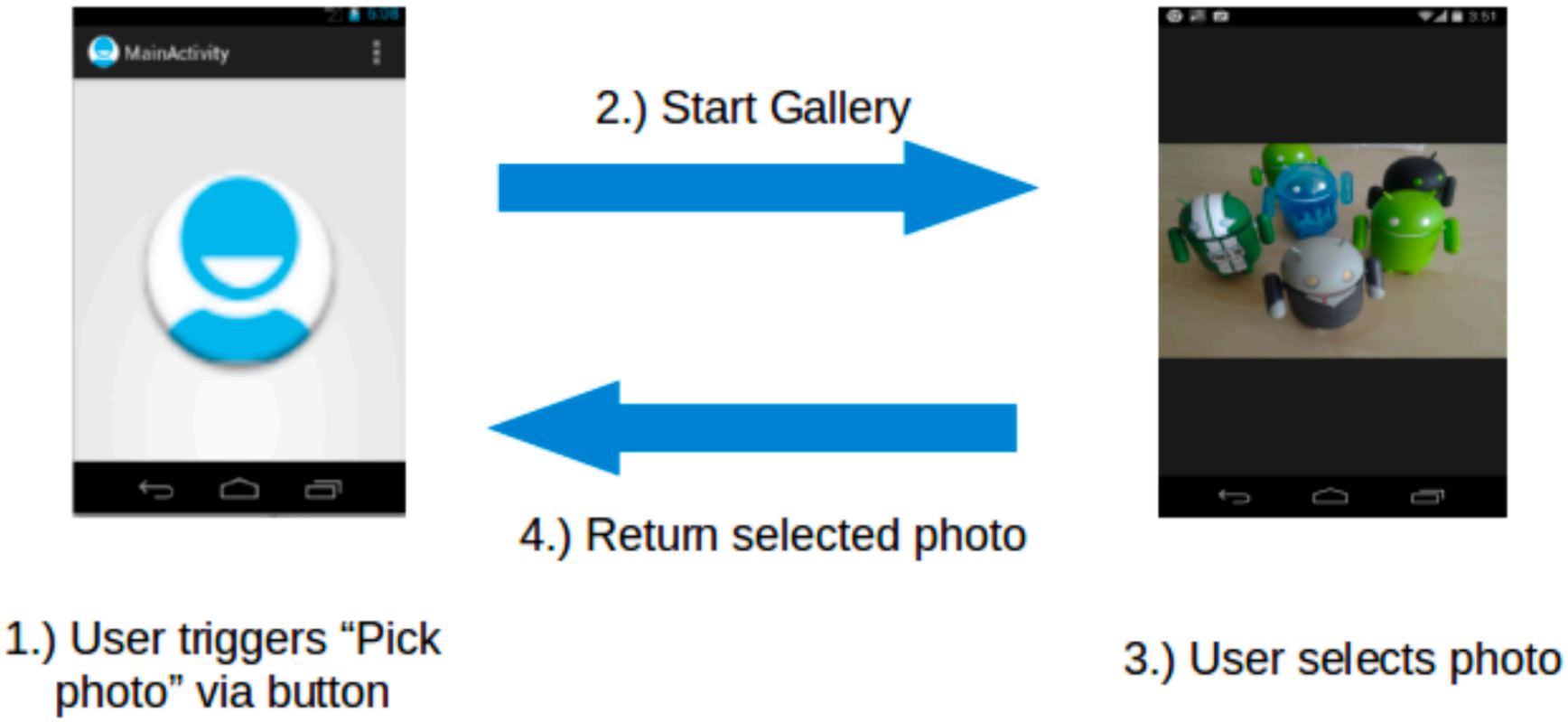
- The first app that sets a permission's protection level wins
  - Later apps can't change it
  - A malicious app that defines a permission first can downgrade its permission level, for example to **normal**
  - Fixed in Android 5.0
    - Links Ch 7e, 7f

Attacking Application Components (to p. 271)

Intents

- ***Intent*** is a data object that defines a task to be performed
- To start an activity, call **startActivity(Intent)**
- **sendBroadcast(Intent)** sends to a broadcast receiver
- **startService(Intent)** sends to a service
- Intent is generic, does not specify tye type of component receiving it

Example



- Link Ch 7g

The following code demonstrates how you can start another activity via an intent.

```
# Start the activity connect to the
# specified class

Intent i = new Intent(this, ActivityTwo.class);
startActivity(i);
```

Explicit Intents

- State the component that must receive it
- Using **setComponent()** or **setClass()**
- Bypasses the intent resolution process in the OS
- Directly delivers the intent to the specified component

Implicit Intent

- Does not specify the component to be used
- Relies on the OS to determine the best candidate to deliver it to
- Ex: "Play this MP3"
  - Using whatever player is available
  - A box may pop up asking the user which app to use

Example

- This intent tells the Android system to display a webpage
- All installed Web browsers should be registered to via an ***intent filter***

```
Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.vogella.com"));
startActivity(i);
```

Intent Filters

- Defined by installed apps
- Filters can match
  - **Action**
  - **Data**
  - **Category**
- **Action** is mandatory

Example Intent Filters

- **Scheme—This is the scheme of any URI. For example, on `https://www.google.com`, the scheme is *https*.**
- **Host—This is the host portion of a URI. For example, on `https://www.google.com`, the host is *www.google.com*.**

Example Intent Filters

- **Port—This is the port portion of a URI. This can catch URIs that target a specific port.**
- **Path, pathPrefix, and pathPattern—These can be used to match any part of the data to a desired value.**
- **MimeType—This defines a specific MIME type for the data that is specified inside the intent.**

am: Activity Manager

- Part of Android
- Lets you send intents to app components
  - Link Ch 7a

```
adb shell am start -a android.intent.action.VIEW
```

Table 2. Available activity manager commands

Command	Description
`start [options] intent`	Start an `Activity` specified by *`intent`*. See the Specification for intent arguments. Options are: • **`-D`**: Enable debugging. • **`-W`**: Wait for launch to complete. • **`--start-profiler file`**: Start profiler and send results to *`file`*. • **`-P file`**: Like **`--start-profiler`**, but profiling stops when the app goes idle. • **`-R count`**: Repeat the activity launch *`count`* times. Prior to each repeat, the top activity will be finished. • **`-S`**: Force stop the target app before starting the activity. • **`--opengl-trace`**: Enable tracing of OpenGL functions. • **`--user user_id | current`**: Specify which user to run as; if not specified, then run as the current user.

M 501: Drozer

- Skip for now, being updated to eliminate drozer

Real-World Examples

Figure 7.4 Device lock screen requiring a password and then this being removed after the exploit is run

Lock Screen Bypass

- On Android < version 4.4
- Sending this intent would disable the lock screen mechanism
- adb shell am start -n com.android.settings/com.android.settings.ChooseLockGeneric --ez confirm_credentials false --ei lockscreen.password_type 0 --activity-clear-task

Tapjacking

- Malicious app overlays a false UI on top of buttons
- So taps activate something unexpected
- Using *toasts* --small graphic elements

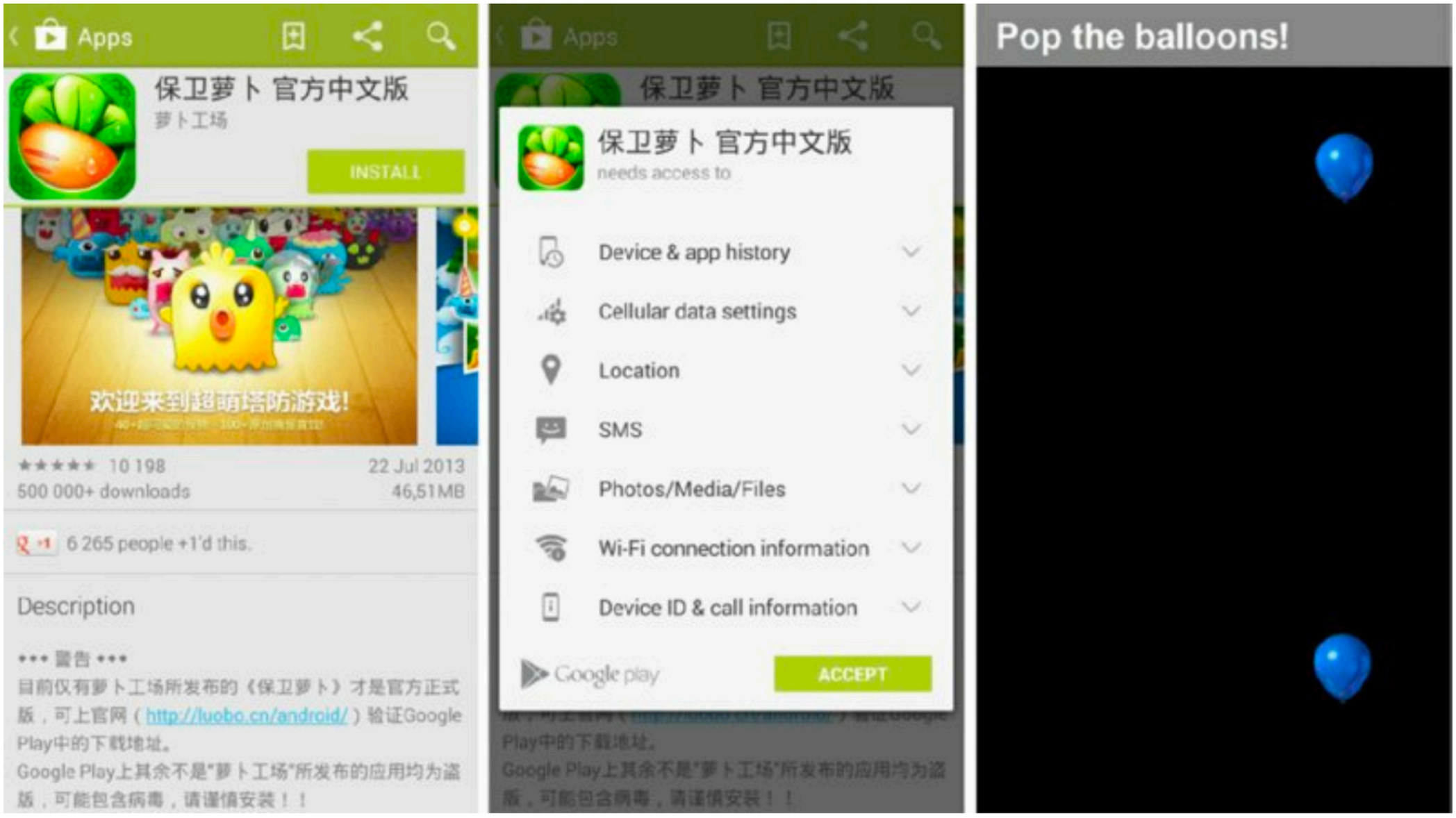
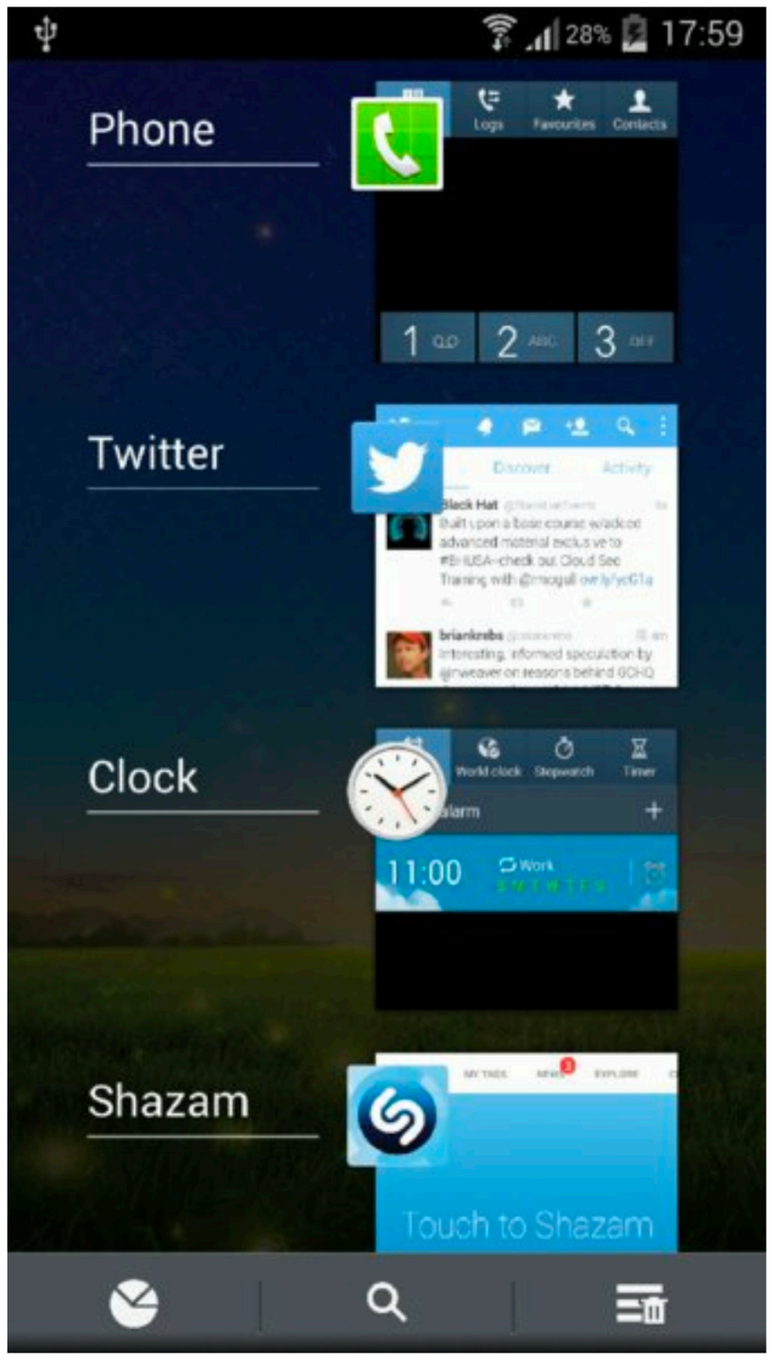


Figure 7.5 An illustration of how a toast could be used to perform unintended actions on underlying activities

Recently-Used App Screenshots

- May contain sensitive info
- Stored in RAM
- Only available to privileged users

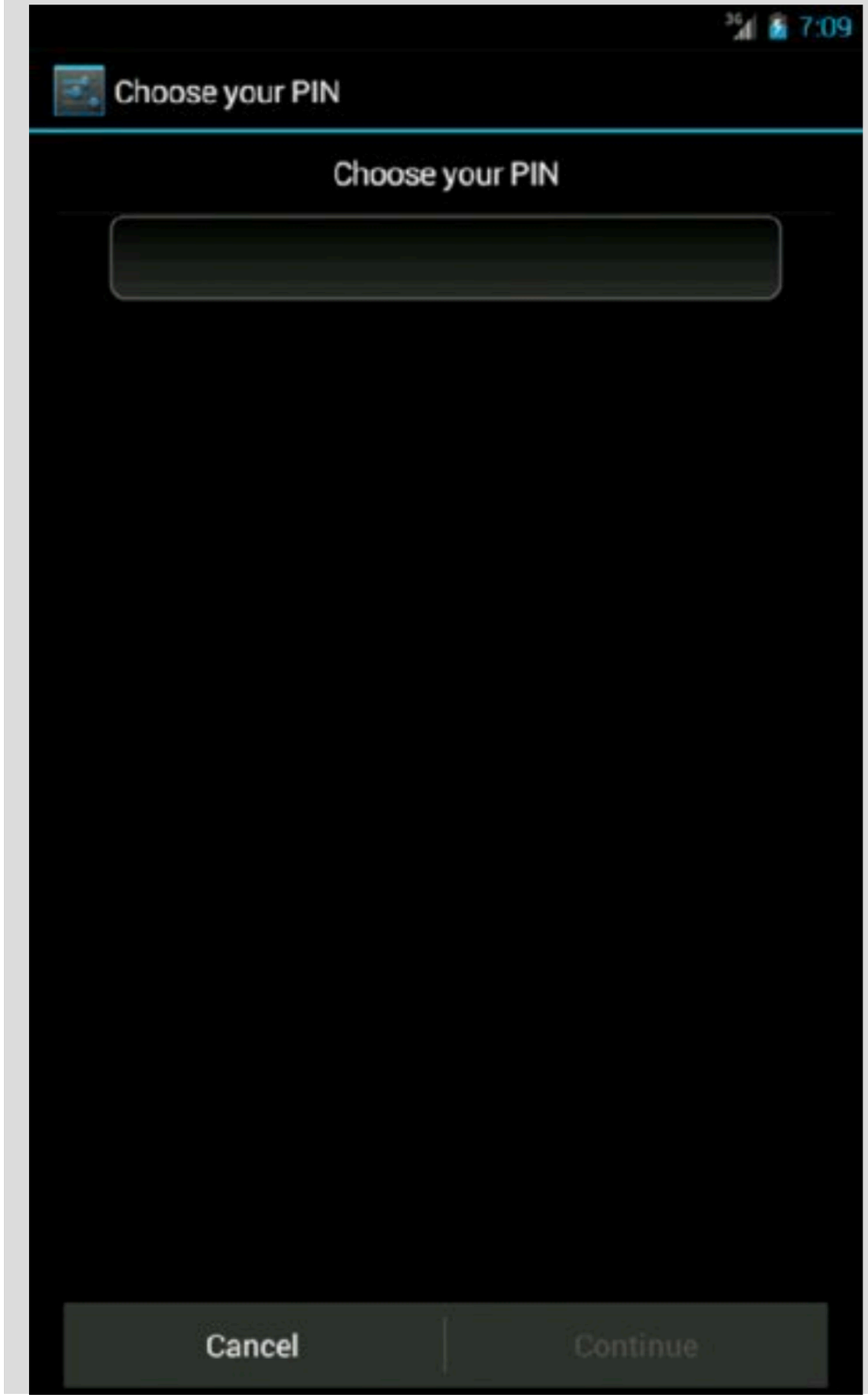


Fragment Injection

- On Android 4.3 and earlier
- Using a "fragment", could change PIN without knowing old PIN

```
dz> run app.activity.start --component com.android.settings

com.android.settings.Settings --extra

string :android:show_fragment

com.android.settings.ChooseLockPassword

$ChooseLockPasswordFragment --extra boolean

confirmcredentials false
```

Opens this screen directly




Kahoot!