Intent on being a good Android citizen?

Defensive strategies and techniques for developers

OWASP
The Open Web Application Security Project

About Me

- Andrew Lee-Thorp
- Security consultant at @**cigital** (UK)
- > 10 years cutting code (smartcard, STB, distributed)
  - @Cigital - Android assessment team (UK), tool development, large scale enterprise design and dev, bug hunting (C/C++), assessing (in)secure architectures

About you

- You are a **developer** and you want to want to write secure Android apps:
  - Services, Activities, BroadcastReceivers, [not ~~ContentProviders]~~
  - Best-practices and Gotchas
- Not a developer
  - Not going to talk about attacks
  - Create concise remediation guidance

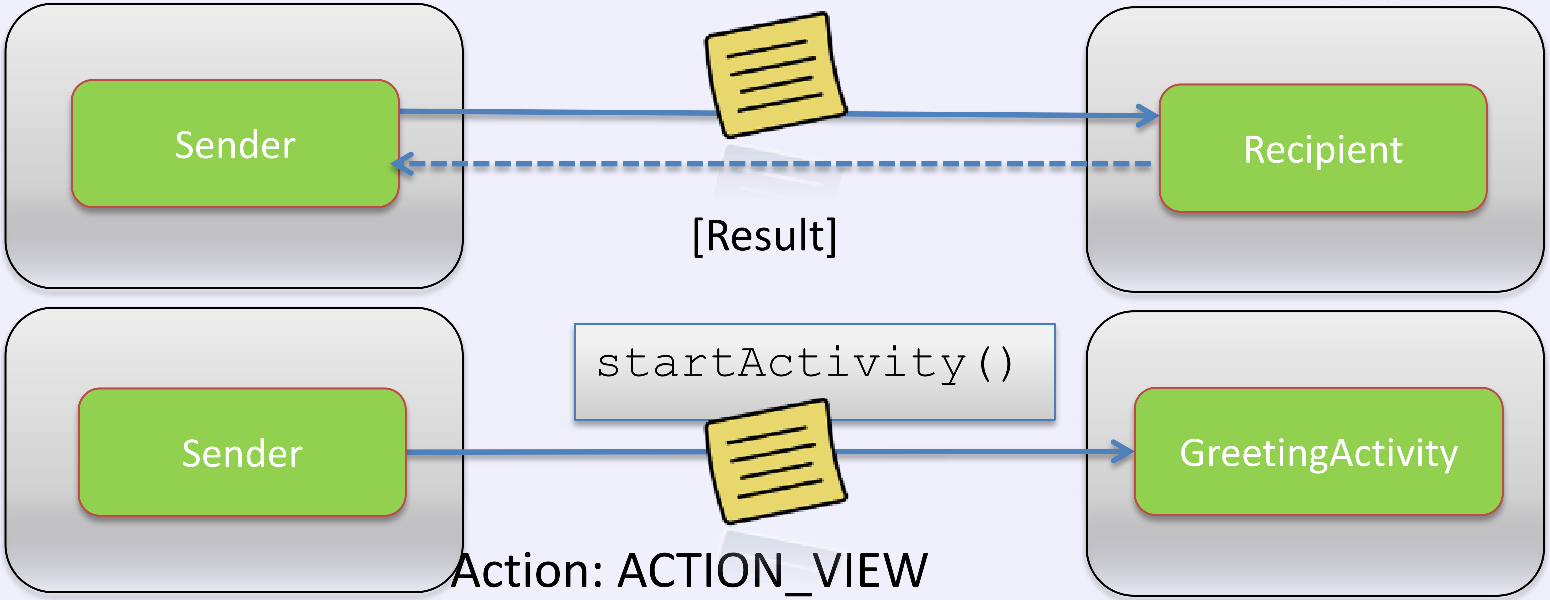
OWASP
The Open Web Application Security Project

Intent Primer

6 rules for safe intents

1. Be explicit about exported.
2. Treat all intent data as evil.
3. Verify intent origin before handling “system” intents
4. Use only explicit intents for internal communications
5. Avoid sending sensitive data in intents
6. Validate your permission assumptions

What is an intent?



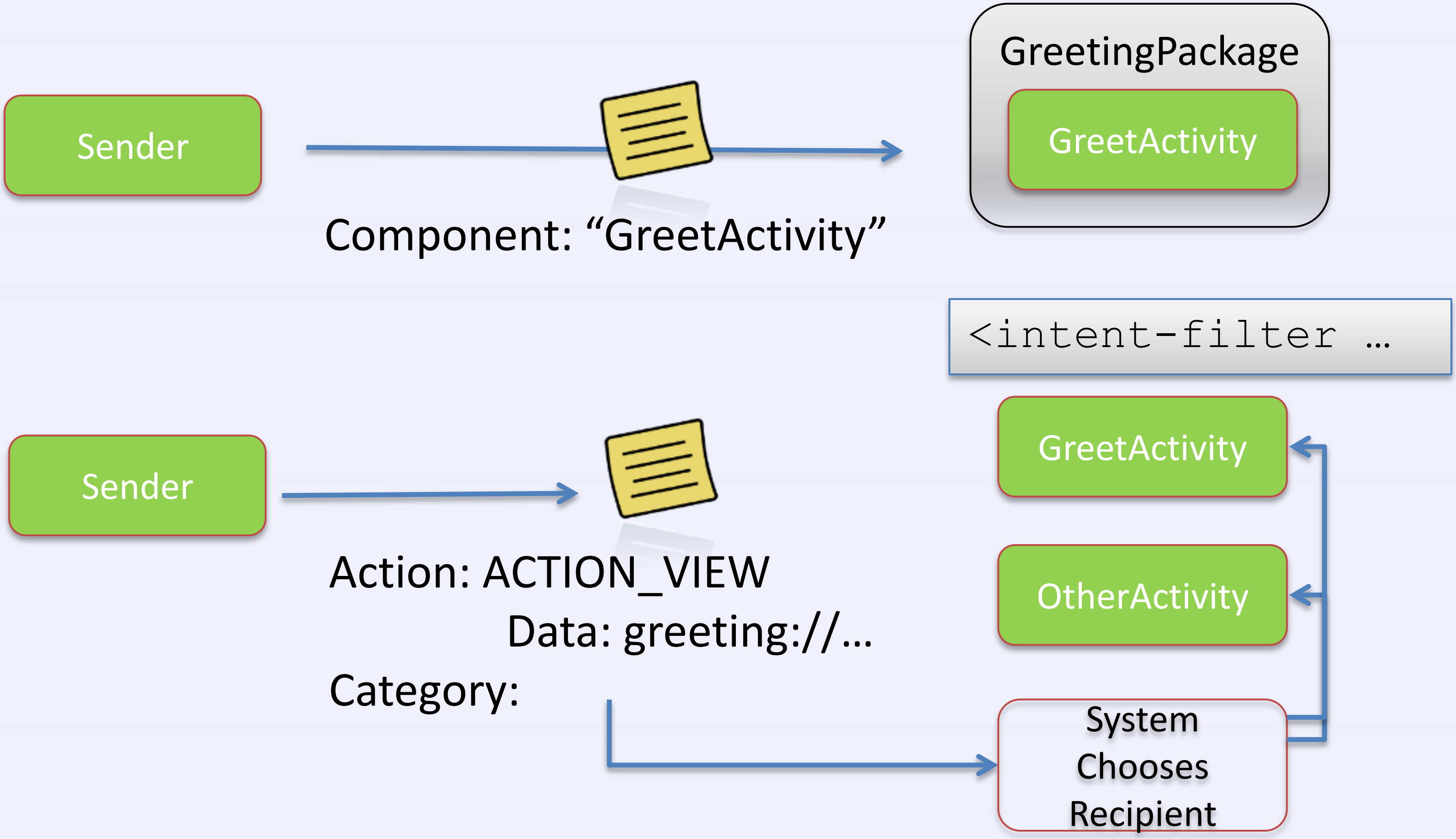
Action: ACTION_VIEW
Data: content://contacts/people/

- Addressing (naming of recipient(s), *)
- Data (send data, optionally receive return value)
- Inter-process mechanism (“RPC”)

Intents can name target or let system decide

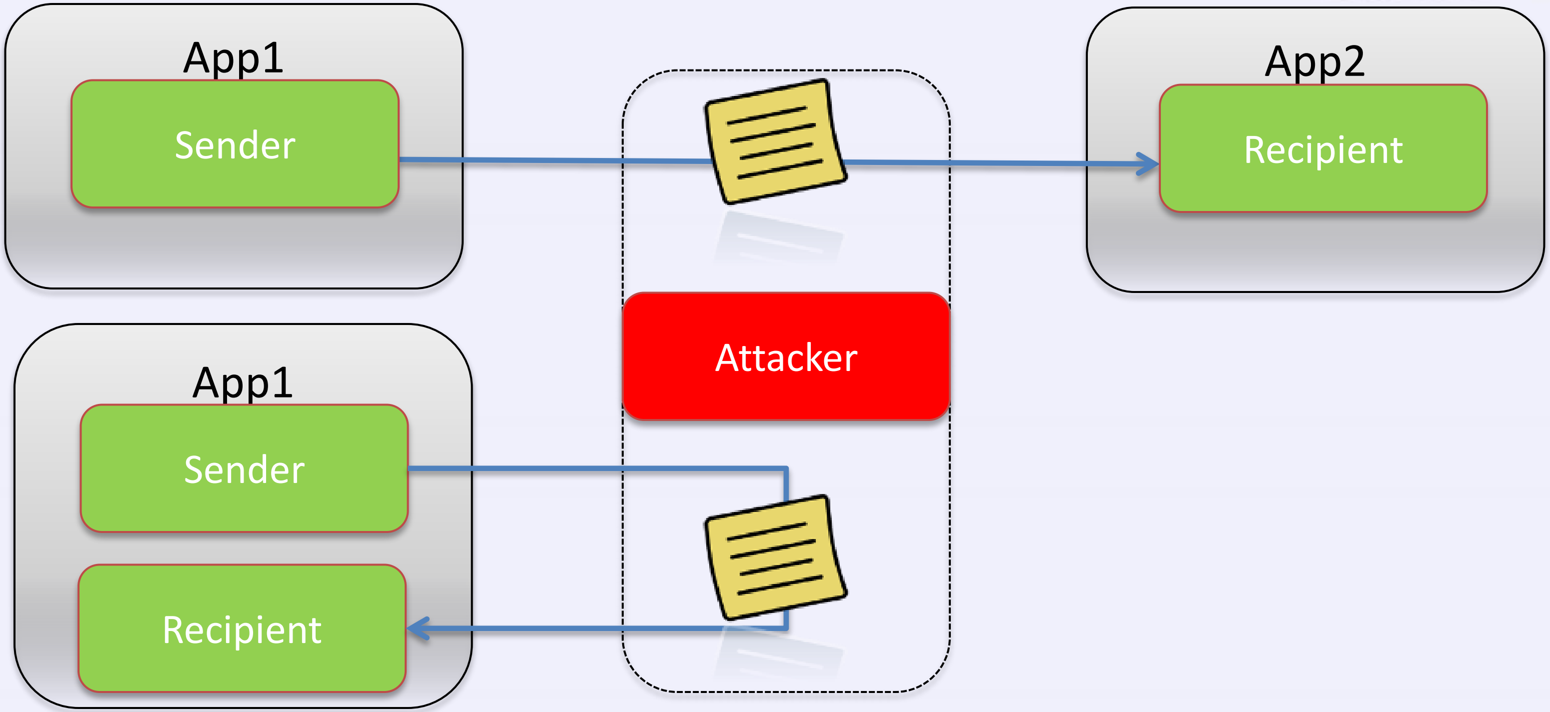
- Explicit versus Implicit

```
intent.setComponent(“GreetingPackage”,“GreetActivity”)
```



```
<intent-filter …
```

Inter-app versus intra-app



- Intents can be used for inter and intra-application communication
- In-process facility exists for broadcasts

Permissions

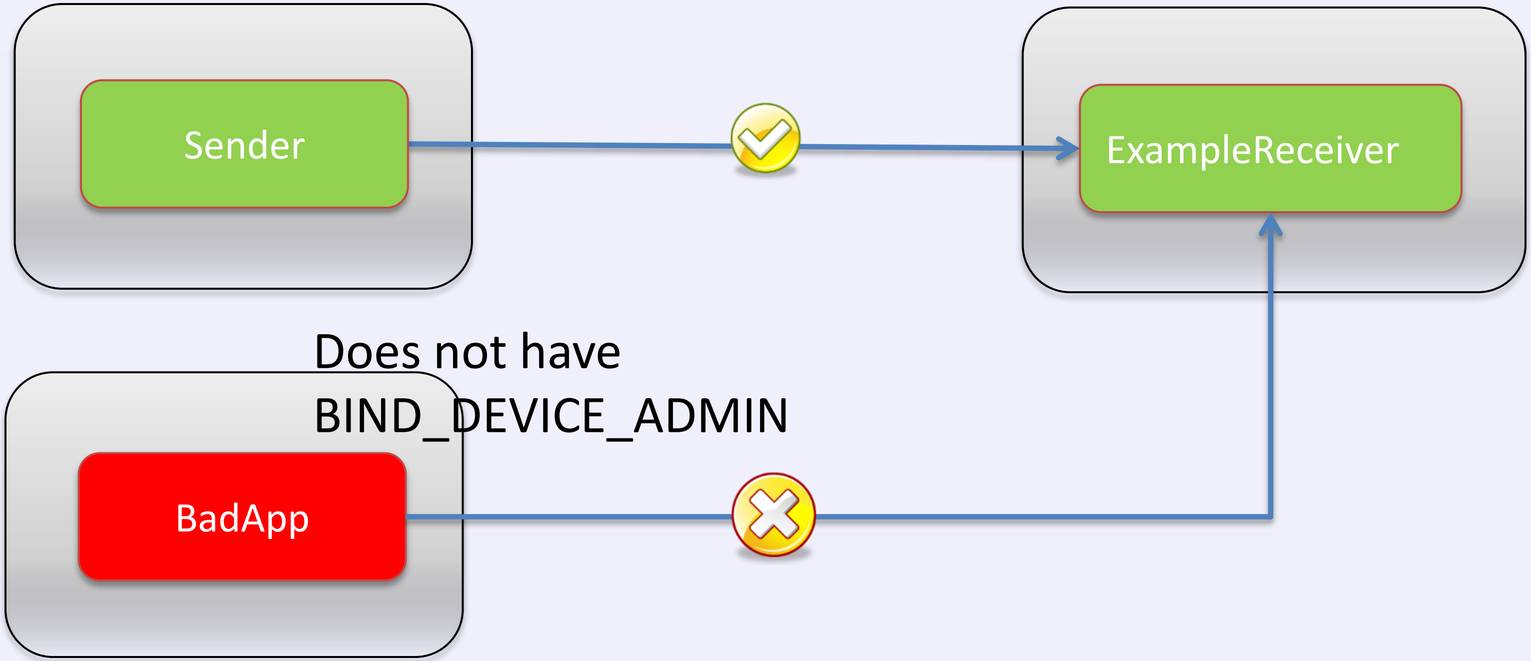
- "Labels" in an access control system used to protect IPC participants
- Built-in and custom
- Protection level determines how permission is granted to requesting app:
  - Normal: annoyance to user, always granted
  - Dangerous: could lead to harm, e.g. cost money, user approved
  - **Signature: granted to requesting apps having same signature**
  - signatureOrSystem: as above and granted to system

Permissions protect receivers

```
<receiver android:”.ExampleReceiver‟
      android:permission=
      "android.permission.BIND_DEVICE_ADMIN">
<intent-filter> …
</receiver>
```

Has BIND_DEVICE_ADMIN

Sender must have BIND_DEVICE_ADMIN



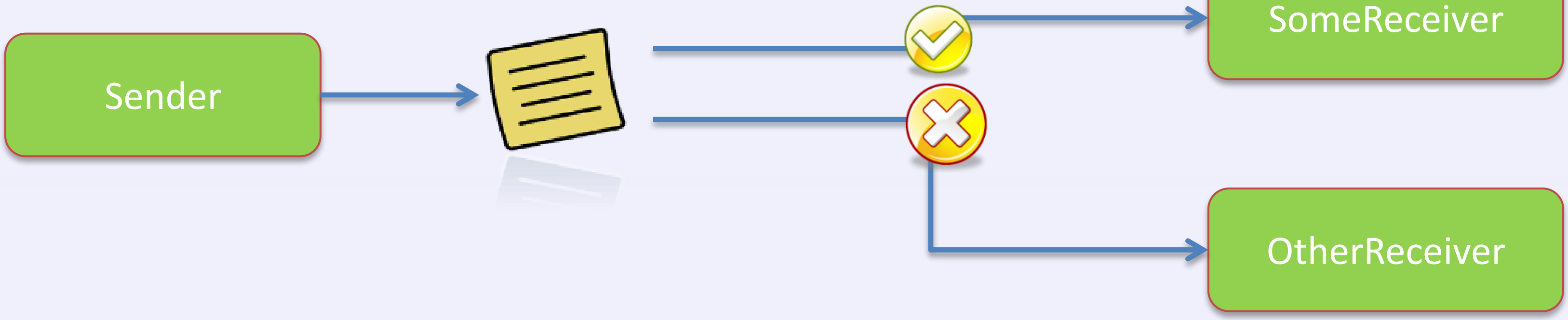
Does not have BIND_DEVICE_ADMIN

Permissions protect broadcast intent senders

```
Intent intent = new Intent();
intent.setAction("com.cigital.SOME_ACTION );
intent.setData(CONTENT_URI);
sendBroadcast(i, "MUST.HAVE.THIS.PERM");
```

Has
MUST.HAVE.THIS.PERM

Recipient must have
MUST.HAVE.THIS.PERM

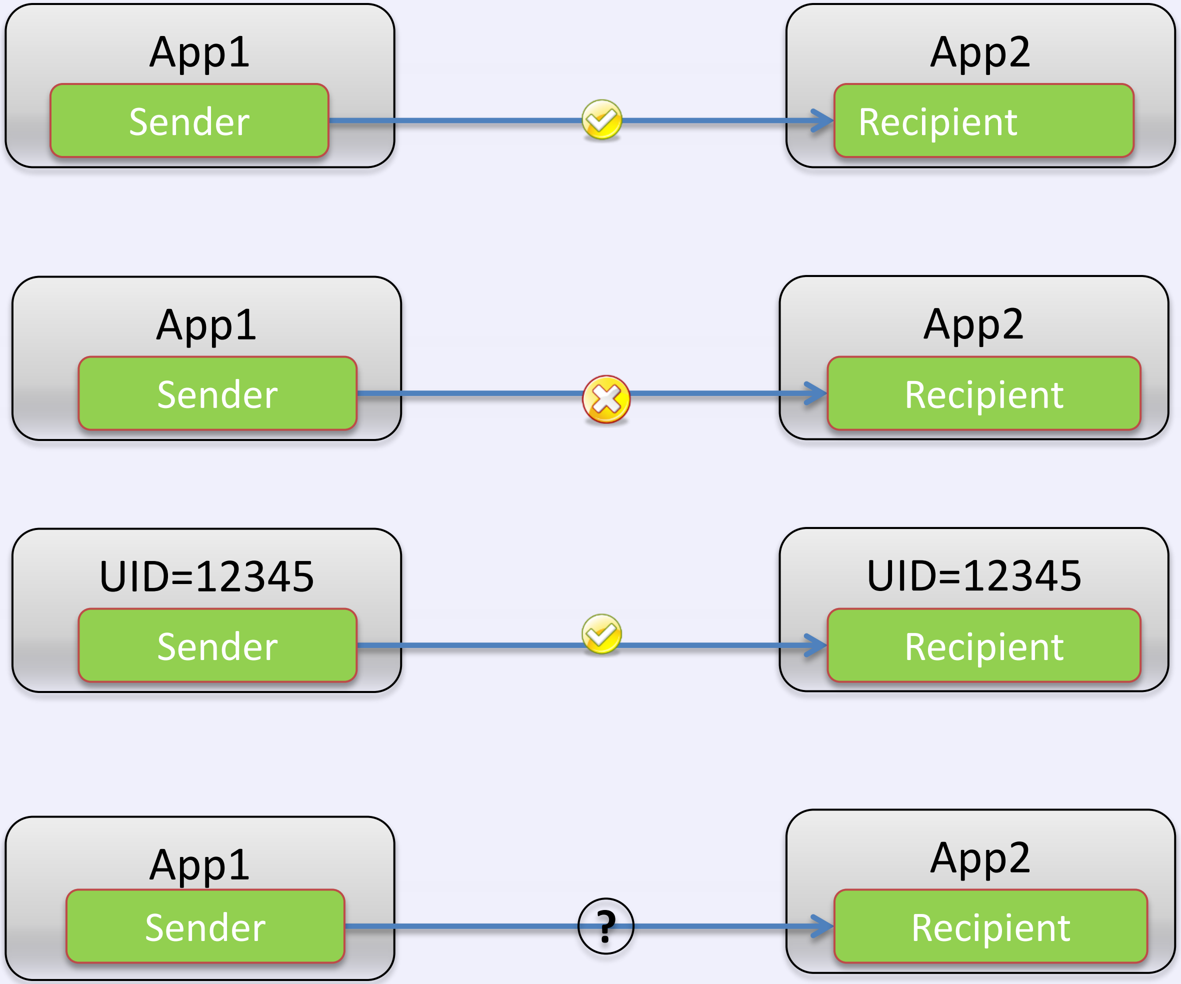


Does not have
MUST.HAVE.THIS.PERM

6 rules for safe intents

1. Be explicit about exported.
2. Treat all intent data as evil.
3. Verify intent origin before handling “system” intents
4. Use only explicit intents for internal communications
5. Avoid sending sensitive data in intents
6. Validate your permission assumptions

1. Be explicit about exported



Public:
android:exported = “true”

Private:
android:exported = “false”

Private:
android:exported = “false”

Question:
android:exported omitted

1. Why be explicit?

- Answer: “it depends”
- Default value is inconsistent.
- exported= false is the idiomatic way of saying “I should only receive intra-app intents”

2. Treat ALL intent input as evil

- For all intent handling code (internal and external)

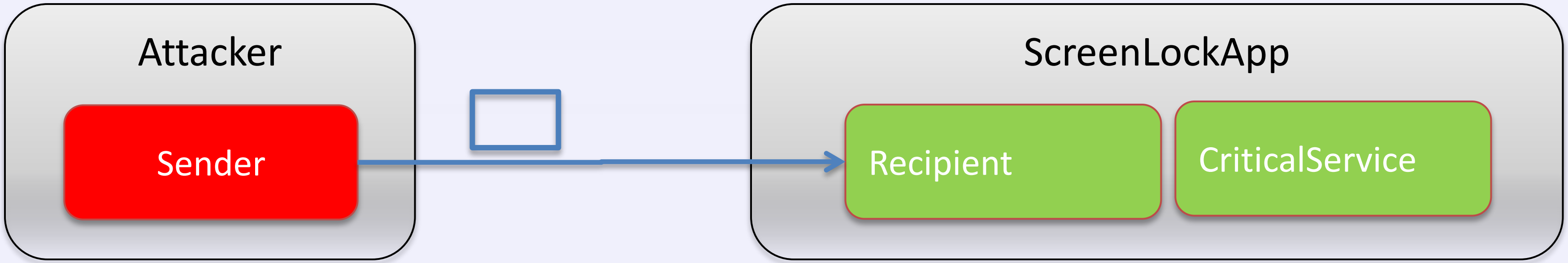
```
String action = intent.getAction();                      1
if (action != null                                       2
      && action.equals(ACTION_I_EXPECT))                 3
String extra =
      intent.getStringExtra(“MY_EXTRA");                 4
if (extra != null                                        5
      && validate(extra)) …
```

- Validate the action

- Validate the extras

2. Trivial Example – empty intents

```
Intent intent =  new Intent();                          1
newIntent.setComponent(…);                              2
startActvity | startService | sendBroadcast             3
(intent)
```



```
String action = intent.getAction();
if (action.equals(…))
      // NullPointerException
      // APP CRASH!!!
      // -> SCREEN LOCK BYPASS
```

2. Example – forgetting to check action

```
<receiver android:name=“MyReceiver1”>  1
 <intent-filter>  2
      <action
      android:name="android.intent.action.PACKAGE_ADDED"  3
```

```
// Forget to check action, just check the extras
if (action != null~~ && action.equals(“PACKAGE_ADDED”)~~){

uid = intent.getStringExtra(“android.intent.extra.UID”);
```

Attacker:

```
Intent intent =  new Intent(“FOOBAR”);
intent.setComponent(“MyReceiver1”); // **
intent.putExtra(“android.intent.extra.UID, attackerUID);
sendBroadcast (intent);
```

3. Verify Intent Origin for (system) broadcast intents

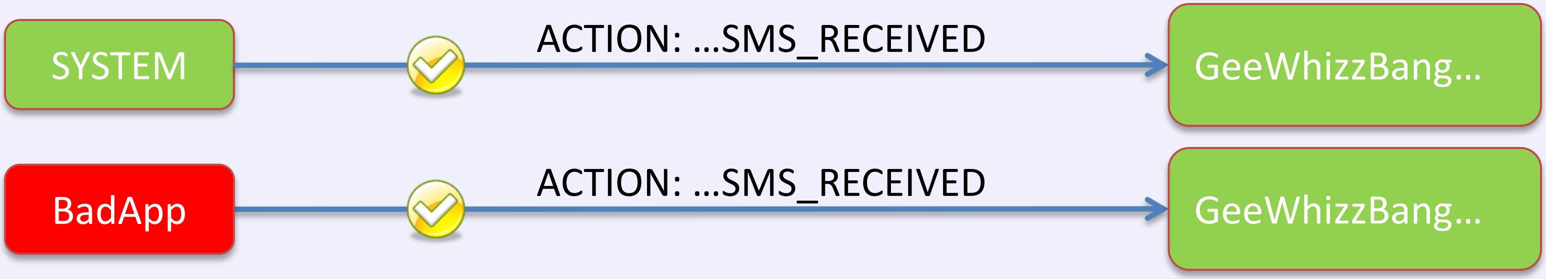
- Two ways:
  - Require sender to have a <permission> that only (system) can have
  - Programatically, e.g.

```
Context.enforceCallingPermission(String permission, String message)
Binder.getCallingUid() == Proces.SYSTEM_UID
```

- Why?
  - Many “system” broadcasts are assumed to be protected
  - Vulnerable to intent spoofing

3. Example

```
<receiver android:name=“GeeWhizzBangMultiFactorAuthReceiver”>
 <intent-filter>
        <action
android:name="android.provider.Telephony.SMS_RECEIVED" />
```



SYSTEM ✓ ACTION: …SMS_RECEIVED → GeeWhizzBang…

BadApp ✓ ACTION: …SMS_RECEIVED → GeeWhizzBang…

```
Protect with:
android:permission="android.permission.BROADCAST_SMS
```

4. Use explicit intents for internal communications**

OWASP
The Open Web Application Security Project

- Use explicit intents for internal communication

```
Intent explicitExmp = new Intent();
explicitExmp.setClassName(this, com.my.specificClass);
startActivity(explicitExmp);
```

- Broadcast Receivers

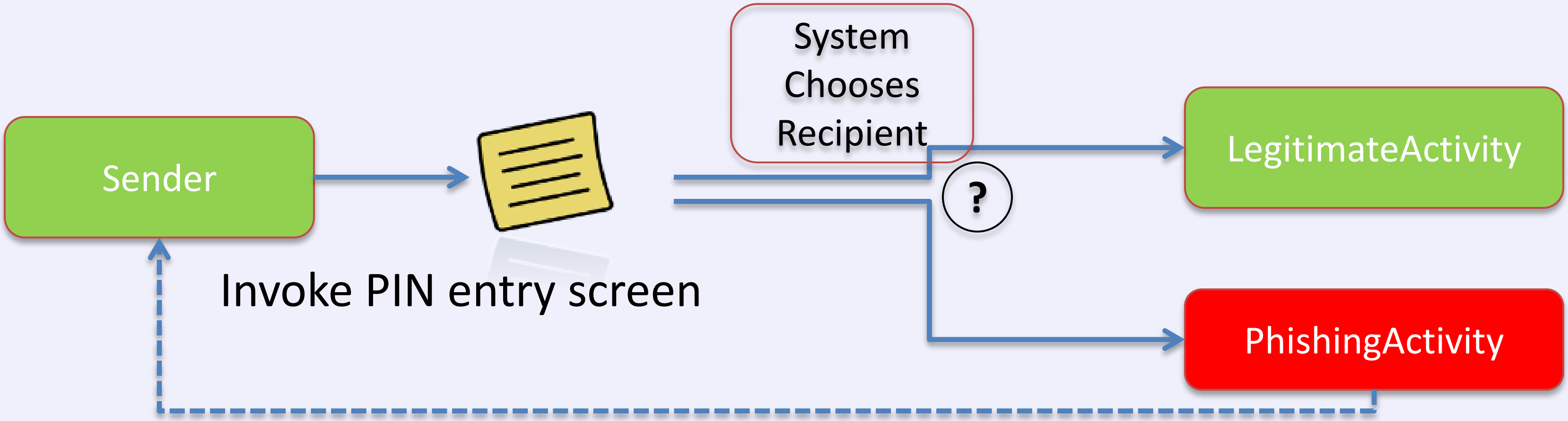
```
// Get a handle to the LocalBroadcastManager then …                     1
localBroadcastManager.registerReceiver(myBroadcastReceiver,
myIntentFilter);

// Elsewhere
localBroadcastManager.sendBroadcast(new                                 2
Intent("com.cigital.MY_ACTION"));
```

4. Why? Unauthorised component can intercept the intent

OWASP
The Open Web Application Security Project

```
Intent implicitExmp = new Intent();
implicitExmp.setAction(android.content.Intent.ACTION_VIEW);
implicitExmp.setData(CONTENT_URI);
startActivity(implicitExmp) | startService | sendBroadcast;
```



```
<intent-filter android:priority=“999”>
         <action android:name="android.intent.action.VIEW"/>
         <data android:mimeType="text/plain"/>
         …
```

Reference: Seven ways to hang yourself with Google Android

4. If you don’t know the addressee name use a strong permission

```
Intent intent =  new Intent( “com.x.y.ACTION", null);                    ❶
intent.addCategory(“COM.X.Y.Z.CATEGORY”);                               ❷
intent.setData(…);

PackageManager mgr = ctx.getPackageManager();
List<ResolveInfo> list =
    mgr.queryIntentActivities(intent, //or queryIntentServices()        ❸
      PackageManager.MATCH_DEFAULT_ONLY);
…
```

```
if(PERMISSION_GRANTED ==
mgr.checkPermission (“SIG.PERMISSION.RCVR.MUST.HAVE”, resolvedPackageName) ❹
{
```

```
         Intent newIntent = new Intent(…
         newIntent.setClassName(resolveInfo.activityInfo.packageName,    ❺
         resolveInfo.activityInfo.name);
         startActvity | startService (newIntent)                         ❻
```

5. Avoid sending sensitive data in intents

```
intent.setData(uriWithSensitiveData);
Intent.putExtra(“extra”, sensitiveData);
// or anything else
// send implicitly or explicitly
```

- Sticky broadcasts leak information
- Service hijacking and subsequent trusted communication
- Phishing
- Intents can leak under circumstances

5. Task information can leak sensitive data

OWASP

The Open Web Application Security Project

Many people familiar with:

```
myhost$ some-program --username=foo --passphrase=bar &
myhost$ ps -a
11234 ?        S       0:00 some-program
myhost$ od -t c /proc/11234/cmdline
0000000   s   o   m   e   -   p   r   o   g   r   a   m  \0   -   -   u
0000020   s   e   r   n   a   m   e   =   f   o   o  \0   -   -   p   a
0000040   s   s   p   h   r   a   s   e   =   b   a   r  \0
```

Same trick in Android

```
// Requires GET_TASKS permission
for (RecentTaskInfo task : activityManager.getRecentTasks(999, 0)) {
        // Access task.origActivity
        // Access task.baseIntent to get the intent data, extras

// Fixed in Android 4.1.1, requires GET_DETAILED_TASKS permission
```

5. Fix

Too many ways it can go wrong that makes it not worth it.

- Instead pass a reference to the data
- Have the recipient fetch the data

6. Validate your permission assumptions on startup

Is that permission really yours?

First (custom) permission registration wins! ☹

```
BadApp installed first:
<permission android:name=“com.goodapp.permission” android:protectionLevel
=         “normal” …>
<uses-permission android:name=“com.goodapp.permission”/>
```

```
GoodApp installed second:
<permission android:name=“com.goodapp.permission” android:protectionLevel
= “signature” ..

<receiver android:name="myReceiver”  android:exported="true"
         android:permission="com.goodapp.permission "> ...
</receiver>
```

6. Validate your permission assumptions

OWASP

The Open Web Application Security Project

- Why is this important?
- Compare **my** declared permissions to **other** declared permissions
- Look at protection level, label, description
- https://github.com/commonsguy/cwac-security/blob/master/security/src/com/commonsware/cwac/security/PermissionUtils.java (there is a bug here)

Summary

1. Be explicit about exported
2. Treat all intent data as evil
3. Verify intent origin before handling “system” intents
4. Use only explicit intents for internal communications
5. Avoid sending sensitive data in intents
6. Validate your permission assumptions

- Questions?

- Thank you!