Information Security Decisions

TechTarget

Android Security Overview

Mike Arpaia
iSEC Partners

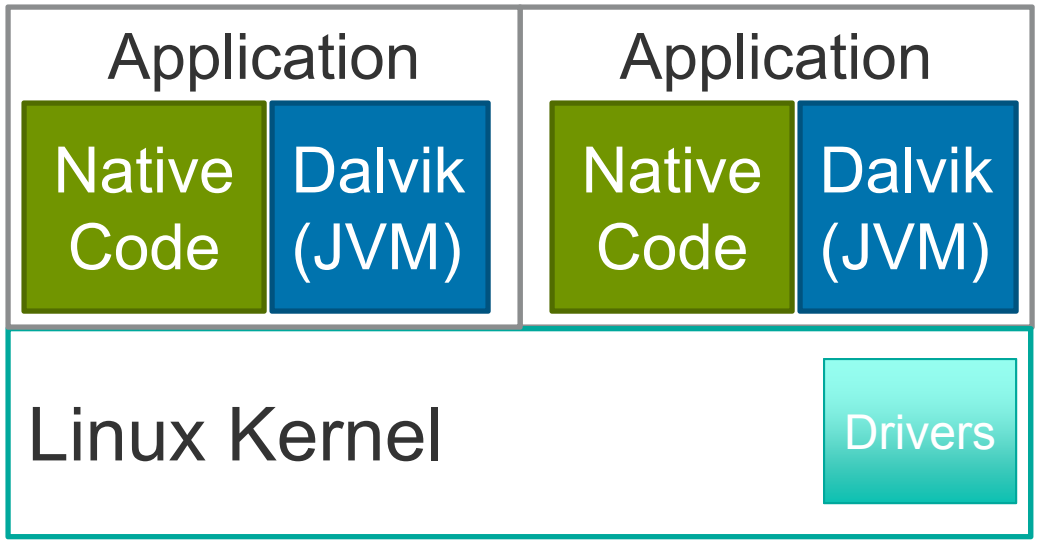
isecpartners
an ncc group company

Agenda

- Background
  - Android applications
  - Android security model
- Application Components – Part 1
  - Intents & Activities
- Application Components – Part 2
  - BroadcastReceivers, Services, and ContentProviders
- Android Gotchas
  - Other issues to worry about

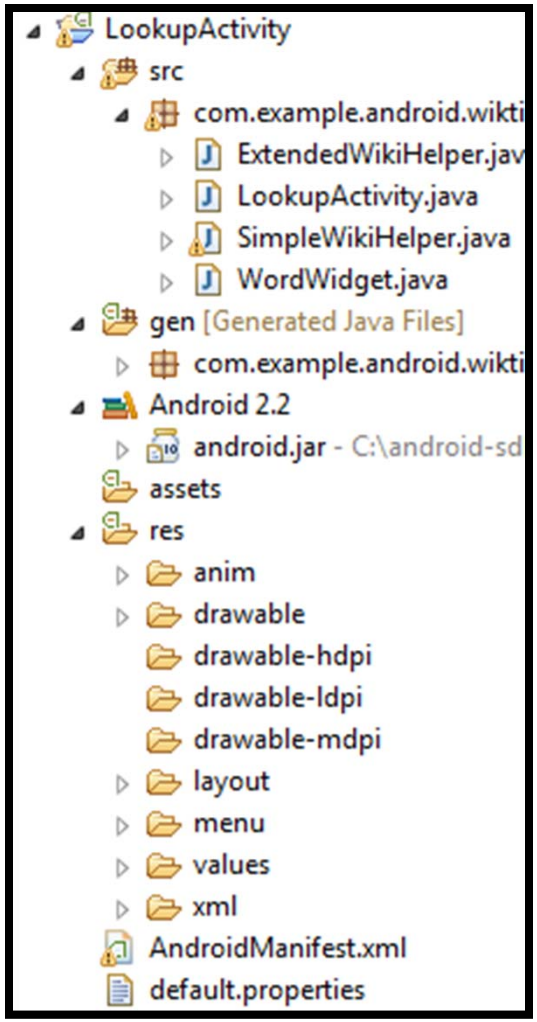
Android Background Intro

- Mobile-optimized Linux distribution
- Introduced in 2008
- “Open” Platform
- 100,000+ Applications



Android App Packaging

- **A**ndroid **P**ac**K**age (.APK)
  - Just a ZIP file (like a JAR)
- Contents listed in manifest
  - AndroidManifest.xml
- Android apps can have both:
  - Java
  - Native (C/C++) code



Android Application Components

- **Activities** – Screens that do something, e.g. the Dialer
- **Services** – Background features, like the IM service
- **Broadcast Receivers** – Actionable notifications (startup!)
- **Content Providers** – Shared relational data
- **Instrumentations** – Rare, useful for testing

Securable with Android Permission:

`"android.permission.READ_CONTACTS"`

or

`"android.permission.BRICK"`

A Sample Android Application

- AndroBuzz – Android client for Google Buzz

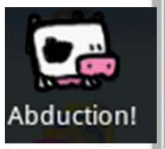
  - Activities
    - BuzzActivity, BuzzNearby, FeedActivity, SettingsActivity, SyncProviderLoginActivity
  - BroadcastReceivers
    - BootLauncher
  - Services
    - AccountAuthenticator, ContactsSync, BuzzService
  - ContentProviders
    - None defined, but uses Android contact database

Android Security Model

- Linux + Android's *Permissions*
- Application isolation
  - Note editor can't read E-Mail
- Distinct UIDs and GIDs assigned on install

cmd - adb shell

```
system   54    31    235472 25044 ffffffff afe0b74c S system_server
bluetooth 78   1     728   172   c00a6164 afe0c69c S /system/bin/hciattach
root     81    2     0     0     c016df24 00000000 D ksdiorqd
root     82    2     0     0     c0058fd4 00000000 S tiwlan_wifi_wq
wifi     85    1     3116  468   ffffffff afe0b874 S /system/bin/wpa_supplicant
bluetooth 94   1     1448  328   c00a6164 afe0c69c S /system/bin/hcid
radio    100   31    140752 13912 ffffffff afe0c824 S com.android.phone
root     174   2     0     0     c0032dc8 00000000 D audmgr_rpc
root     10697 2     0     0     c0175670 00000000 S mmcqd
app_8    17319 31    131380 17068 ffffffff afe0c824 S android.process.acore
root     21488 1     652   136   c0197308 afe0c0bc S /system/bin/debuggerd
root     22824 2     0     0     c0032dc8 00000000 D audmgr_rpc
app_11   22859 31    101844 11280 ffffffff afe0c824 S com.google.process.gapps
shell    25918 38    724   228   c0049ec0 afe0c4cc S /system/bin/sh
app_36   26052 31    109832 19684 ffffffff afe0c824 S com.google.android.voicesearch
app_0    26090 31    99240 14580 ffffffff afe0c824 S com.android.im
app_0    26095 31    94468 12964 ffffffff afe0c824 S android.process.im
app_45   26100 31    96552 13308 ffffffff afe0c824 S au.com.phil
shell    26107 25918 868   328   00000000 afe0b50c R ps
$ _
```



Android Security Model

- Android is not the Java security model:
  - No Java Security Manager, no Java Sandbox
  - Dalvik NOT a security barrier
- This is not the iPhone security model:
  - Platform permissions restrict applications
  - Very open for development & customization
- Closest to OS user isolation, but each app is a user
- Usually supports OTA updates

Android Security Model

- Rights expressed as *Permissions* & Linux groups!

```
cmd - adb shell
C:\>adb shell
$ id
id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input)
,1007(log),1011(adb),3001(net_bt_admin),3002(net_bt),3003(inet)
$
```

```
5:09 PM
$ id
uid=10026(app_26) gid=10026(app_26) grou
ps=3003(inet)
$
```

```
pTerminal
android:/$
uid=10047(app_47) gid=10047(app_47)
```

Permissions

- Based on Linux, UIDs, File permissions
- Each app assigned own user and group

- Permissions granted in the manifest
- Declared at Install time and are static
  - Permission changes during update prompt the user

Manifest Permissions

Browser Permissions

android.permission.

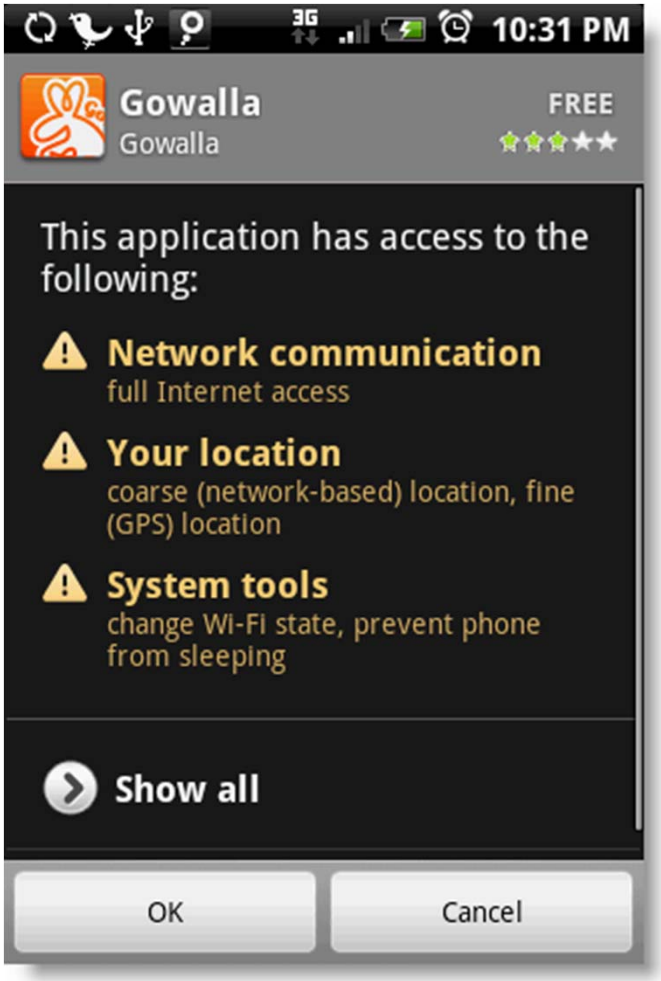
- **.INTERNET**
- **.ACCESS_FINE_LOCATION**
- .ACCESS_COARSE_LOCATION
- .ACCESS_FINE_LOCATION
- .ACCESS_DOWNLOAD_MANAGER
- .ACCESS_NETWORK_STATE
- .ACCESS_WIFI_STATE
- .SET_WALLPAPER
- .WAKE_LOCK
- .WRITE_EXTERNAL_STORAGE
- .SEND_DOWNLOAD_COMPLETED_INTENTS

Twitter Permissions

android.permission.

- **.INTERNET**
- **.ACCESS_FINE_LOCATION**
- .VIBRATE
- .READ_CONTACTS
- .WRITE_CONTACTS
- .GET_ACCOUNTS
- .MANAGE_ACCOUNTS
- .AUTHENTICATE_ACCOUNTS
- .READ_SYNC_SETTINGS
- .WRITE_SYNC_SETTINGS
- .GET_TASKS
- .USE_CREDENTIALS

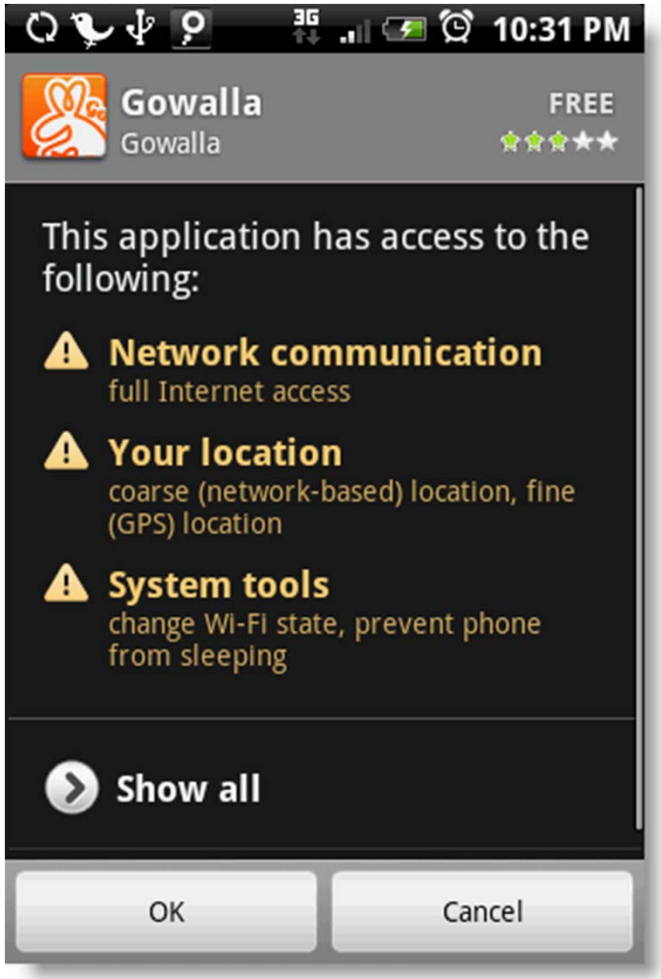
Requesting Permissions



Permission Groups

- Optional, helps display permissions to the user

```
android:permissionGroup=
        "android.permission-group.LOCATION"

android:name=
        "android.permission.ACCESS_COARSE_LOCATION

android:name=
        "android.permission.ACCESS_FINE_LOCATION
```



Application Signing

- Certificates determine identity
  - Set the Application's UID and the GID
- Market can use this to identify trusted developers
  - Identity X has produced good apps for Y years
- Most certificates are self-signed! Not a CA trust model.
- Two applications can share data with:

```
android:sharedUserId="aexp.share.sharedapp"
```

Key Management

- Protect your Android application signing key
- Store on a secure build server
- Audit access and use
- Backup in a secure location

- Protect like a SSL certificate, but perhaps better

Android Background Takeaways

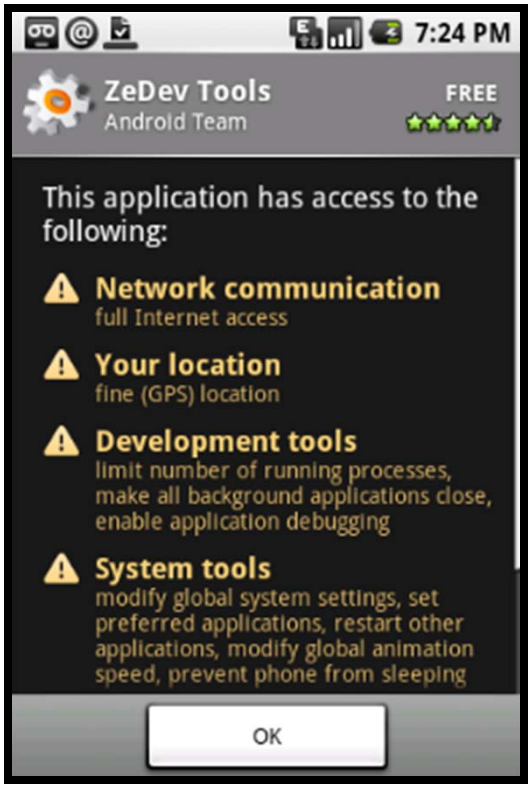
- Mobile optimized Linux distribution
- Android apps are distributed as APKs
  - Similar to jar files, but can contain native code
  - Sandboxed at the OS level
- Rich permission model
  - Obtain access to resources by requesting permissions
  - Permissions are organized into groups
  - Signed to determine identity
  - Most applications can use standard permissions
- Protect your signing certificate!

Android Application Components

- Intents & Activities

Permissions Refresher

```
<manifest xlmns:android...>
   ...
   <uses-permission
android:name="android.permission.INTERNET">
   </uses-permission>
</manifest>
```



Permissions Refresher

Securable Object	Effect
Activity	Who can start the activity?
Service	Who can start, stop or bind to the service?
BroadcastReceiver	Who can send broadcasts to the receiver? Rights needed by the receiver of a broadcast
ContentProvider	Who can access data in the ContentProvider?

Defining New Permissions

- Exposing a service to other applications
- Frequently accessed, dangerous
- Difficult for users to permit
- Want to go “on the record” about what apps expose

- New permissions are rare

Custom Permissions

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.me.app.myapp" >
  <permission
    android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
    android:label="@string/permlab_deadlyActivity"
    android:description="@string/permdesc_deadlyActivity"
    android:permissionGroup="android.permission-group.COST_MONEY"
    android:protectionLevel="dangerous" />
</manifest>
```

- **name:** The permission's name used in code
- **label:** The localizable name shown to the user
- **description:** Description text shown to the user
- **permissionGroup:** A group of permissions to associate this permission with
- **protectionLevel:** Determines how to prompt the user (normal & dangerous)

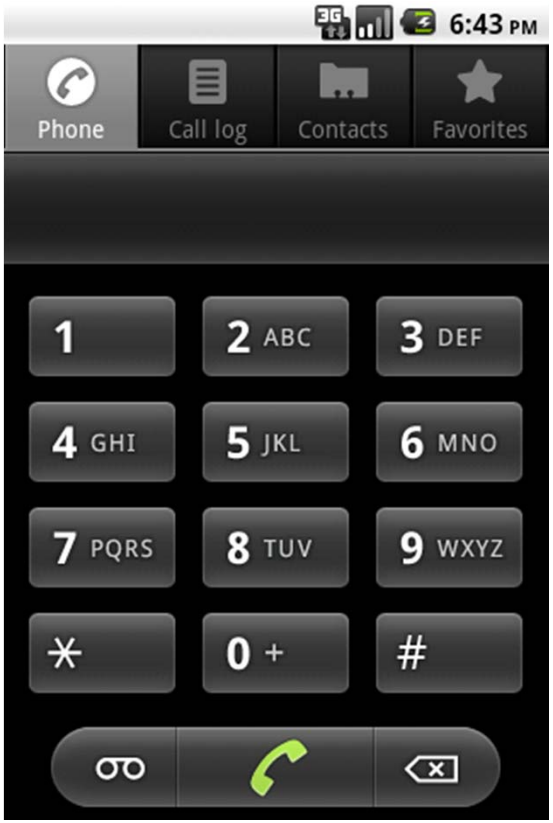
Intents: Android IPC Messages

- Used for Activities, Broadcasts, Services, and More

Contacts App

User Clicks

Sends Intent



Dialer App

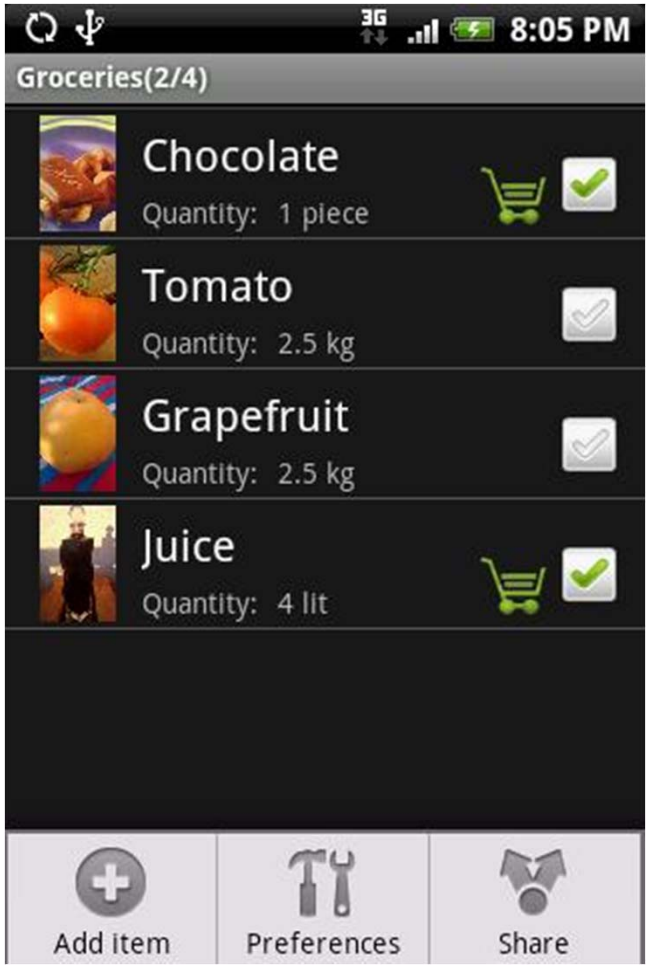
Intents As Weapons

- Intents are used every Android application
- All applications can send intents
  - Even malicious ones!
- Intents carry data
  - Data can be malicious
  - Your app could leak data!
- Must handle malicious intents
  - Or use permissions to restrict who can send them to you

Activities

- “An activity is a single thing that the user can do.”
- Example activities:
  - ATM locator screen
  - Dialer interface
  - Foursquare “checkin” page
- Can receive intents
  - E.g. a Dialer intent with a phone #

Protecting Activities



Does your Activity perform actions on behalf of the user?

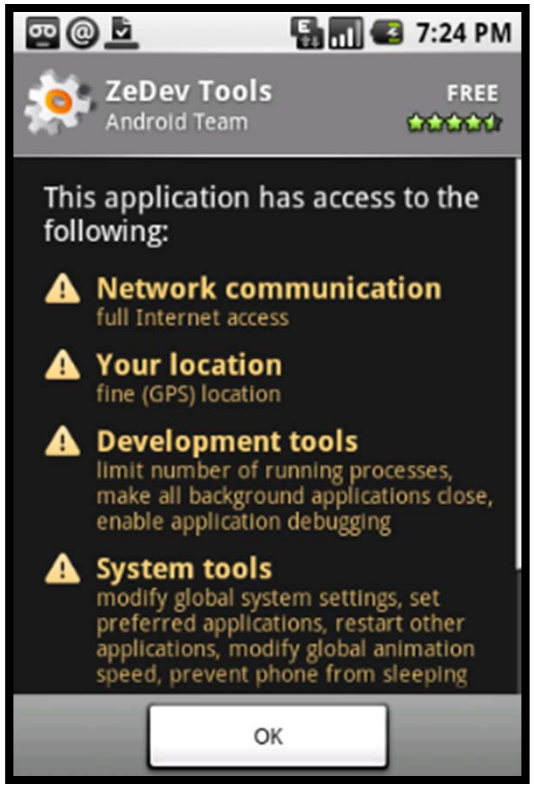
IntentFilters: Not Authoritative

```
// The browser's intent filter isn't interested in this action
Intent i = new Intent("Cat-Farm Aardvark Pidgen");

// The browser's intent filter isn't interested in this Uri scheme
i.setData(Uri.parse("marshmaellow:potatochip?"));

// The browser activity is going to get it anyway!
i.setComponent(new ComponentName("com.android.browser",
                        "com.android.browser.BrowserActivity"));
```

Two Way to Secure Activities

Confirm (every time)

Permission (applied once)



How to Avoid Custom Permissions

- Custom permissions can be clumsy

- Instead:
  1. Start an activity
  2. Confirm the action with the user

- Example:
  1. Dialer application launches
  2. Shows # and asks user to dial
  3. User must confirm before dial

How to Get Confirmation

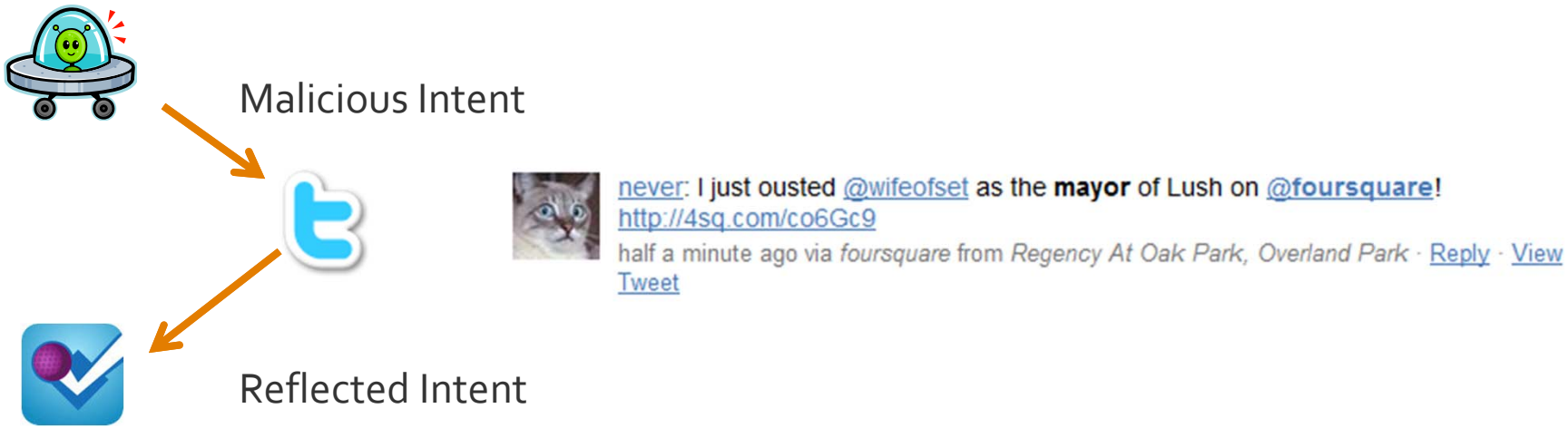
```
AlertDialog.Builder builder = new
   AlertDialog.Builder(this);
builder.setMessage("Do you want to self-destruct?")
   .setCancelable(false);

builder.setPositiveButton("Yes", new
   DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int id) {
    MyActivity.this.finish();   }
});

builder.setNegativeButton("No", new
   DialogInterface.OnClickListener() {
   public void onClick(DialogInterface dialog, int id) {
     dialog.cancel();
    }
});

AlertDialog alert = builder.create();
```

How to Apply Activity Permissions

```
<activity
  android:name=".BlankShoppingList"
  android:permission =
     "com.isecpartners.ACCESS_SHOPPING_LIST">
<intent-filter>
  <action
  android:name=
   "com.isecpartners.shopping.CLEAR_LIST" />
</intent-filter>
</activity>
```

Intent Reflection

- Don’t let malicious apps push you around!



- PendingIntents store the identity of the original caller

Android App Component Takeaways

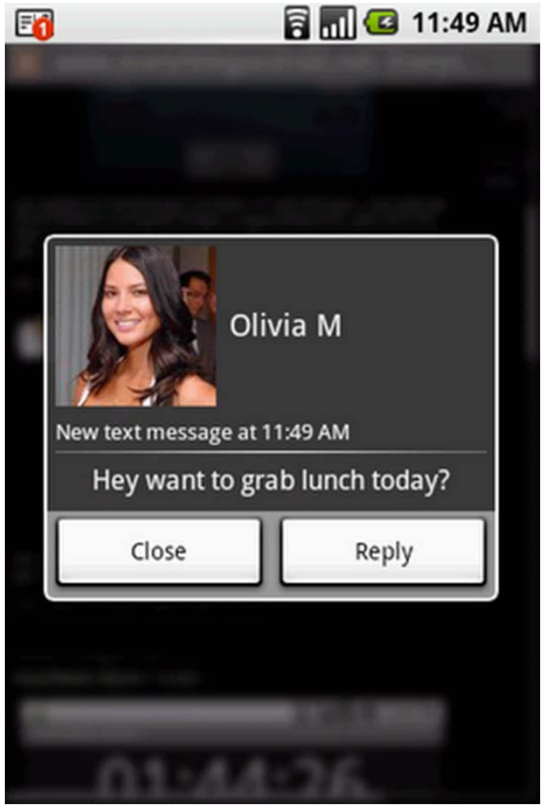
- Android components communicate via Intents
- Intents can be malicious & must be handled with care
- Be careful what activities you support
- Permissions can restrict who can send intents

Android Application Components

- BroadcastRecievers, Services, and ContentProviders

BroadcastReceivers

- Let apps and system communicate via intents
- Android handles:
  - dispatching,
  - starting receivers,
  - and enforcing permissions
- Broadcasts may be malicious
- Apps could sniff broadcasts



Protecting BroadcastReceivers

- Don't export if possible
- Set permissions on send and on receive
- For receive (who can send Intents to me):

```
<receiver android:enabled="true"
  android:exported="false"
  android:name="com.isecpartners.Sample"
  android:permission="android.permission.RECEIVE_MMS">
</receiver>
```

- For Send (who can receive my broadcasts):

```
Context().sendBroadcast(intentObject,
       "android.permission.RECEIVE_MMS“);
```

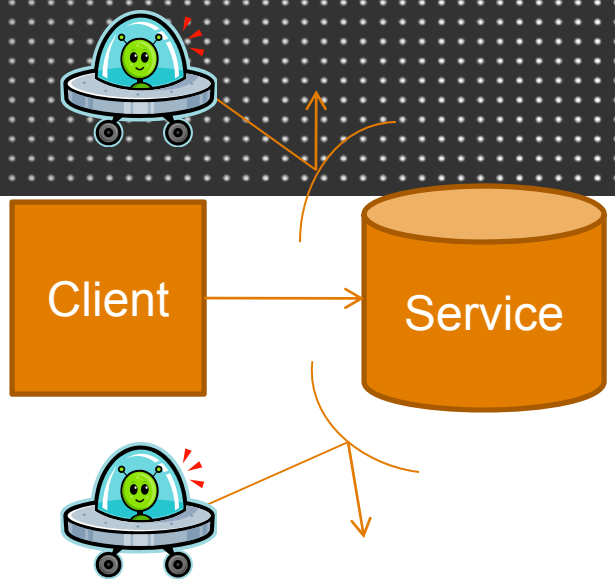
Sticky Broadcasts

- Sticky Broadcasts are usually informational
  - For example, system state like the battery
- You can't apply permissions to them

```
Intent intent = new Intent("com.bank.checkcleared");
intent.putExtra("Check 01234", true);
sendStickyBroadcast(intent); // everyone can read me!
```

Don't use StickyBroadcasts for exchanging information

Services



- Background Processes
- Sample Services:
  - Schedule MP3s
  - Store passwords or Private Messages
  - Retrieve e-mail periodically
- Permissions can apply to services

Service Mutual Authentication

- Be careful when sending sensitive data to a service
  - E.g. Passwords, Credentials
- Must check the service's identity

How to Authenticate Services

- Option 1: Specify the service explicitly in the Intent

```
Intent svc = new Intent(AndroBuzzActivity.this,
                        AndroBuzzService.class);
startService(svc);
```

- Option 2: Verify against name provided by onServiceConnected event

- Option 3: Use the component name to validate permissions (to dynamically allow replacement services)

```
res = getPackageManager().
      checkPermission(permToCheck, name.getPackageName());
```

ContentProviders

- SQL databases that store text, images, sounds…
- Permissions determine who can read or write
  - Caveat: Anyone with write effectively has read access

```
<provider android:authorities="list"
          android:enabled=true
          android:exported=false
          android:grantUriPermissions=["true" | "false"]
          android:name="string"
          android:permission="string"
          android:process="string"
          android:readPermission="string"
          android:syncable=["true" | "false"]
          android:writePermission="string" >
    . . .
</provider>
```

Querying ContentProviders

- Use a URI
  - "content://com.example.travel/trains/122"
    - “Give the me the train with ID #122”
  - Tables can have sub-tables
    - "content://com.example.travel/trains/baltimore"
- Don’t do this:
  - "content://com.example.travel/trains/" **+ id**
  - What if user controlled id and accessed a sub-table?
- Use:
  - ContentUris.withAppendedId()

Android Component Takeaways

- Permission BroadcastReceivers
- Permission broadcast intents
- Do not use private data in sticky intents
- Keep ContentProviders private
  - Use permissions for exported providers
  - Careful when assembling URIs
- Mutually authenticate services

Android Component Summary

- Lots of different attack surfaces to watch
- Export a small attack surface
- Be aware of:
  - Where you **send** intents
  - Where you **receive** intents from
- Use pre-defined permissions if possible
  - May not be granular enough

Android “Gotchas”

- Specific issues to watch out for

Access Level Modifiers Don't Work

- We see **@hide** on classes, or individual methods

```
/**
 * @hide Broadcast intent when the volume for a particular stream type changes.
 * Includes the stream and the new volume
 *
 * @see #EXTRA_VOLUME_STREAM_TYPE
 * @see #EXTRA_VOLUME_STREAM_VALUE
 */
 @SdkConstant(SdkConstantType.BROADCAST_INTENT_ACTION)
 public static final String VOLUME_CHANGED_ACTION = "android.media.VOLUME_CHANGED_ACTION";
```

- NOT a security boundary, trivially bypassed

Storing Data Locally

- Don’t use external storage, it’s FAT32

  ~~`GetExternalFilesDir()`~~

- External storage is readable by all processes
  - Write requires permission as of Donut

- Avoid storing data locally!

Storing Data Locally

- Use internal storage, it has strong permissions

```
String FILENAME = “pubkey";
String string = “---BEGIN PUBLIC KEY---…";

FileOutputStream fos = openFileOutput(FILENAME,
                                      Context.MODE_PRIVATE);

fos.write(string.getBytes());
fos.close();
```

Avoiding Cache Issues

- Embed the “WebView” control carefully
- When working with sensitive pages:

```
WebView.WebSettings.setSaveFormData() = False;
```

- Set Cache-Control HTTP Headers:

```
Cache-Control: no-cache no-store
```

Creating SSL Connections

- Use SSL, for *everything*
- Default HTTPS Class Checks:

✔ Name
✔ Expiration
✔ Issuance
✘ Revocation

- Sample:

```
URL url = new URL("https://www.isecpartners.com");
URLConnection urlConn = url.openConnection();
```

Native Code

- Good for games; avoid otherwise
- Subject to standard C language issues
- Still running as the application's UID
- Avoid if at all possible

Gotcha Takeaways

- Careful when using internal storage
- Control caching
- Default HTTPS class performs proper certificate checks
- Avoid native code

Android Summary

Android Takeaways

- Rich security model
- Robust IPC mechanism
- Potential for large attack surface

Android Secure Coding Checklist

- Use least privilege
- Do not unnecessarily export components
- Handle intents carefully
- Justify any custom permissions
- Use PendingIntents to protect against Intent Reflection
- Mutually authenticate services
- Use APIs to construct ContentProvider URIs
- Watch WebView caching
- Avoid Native Code (and review what you write)
- Use HTTPS
- Store very little data locally

Mike Arpaia

mike@isecpartners.com

iSEC Partners

Featured Member of the TechTarget Editorial Speaker Bureau