Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Understanding Android's Security Framework

William Enck and Patrick McDaniel

Tutorial

October 2008

Telecommunications Nets.

- The telecommunications industry is in a period of transition to open handsets, open/augmented services and high-speed data.
  - Openmoko
  - LiMo
  - Android

- *Idea*: open platform to application developer ingenuity, will lead to new market drivers and differentiators.

This tutorial

- *We are here to describe the security available in Android*.
- Goals
  - Broadly catalog Android's application architecture
  - Describe how security policy is defined within Android
  - Describe the interfaces used to define policy
  - Best practices for using those interfaces
  - Show some pitfalls leading to insecure applications

- We *not* here to teach you to build Android apps ...
- Follow along at http://siis.cse.psu.edu/android_sec_tutorial.html

What is Android?

- One of the most anticipated smartphone operating systems -- led by Google
  - Complete software stack
  - Open source (Apache v2 license) ... mostly

- Open Handset Alliance
  ... 30+ industrial partners
  - Google, T-Mobile, Sprint, HTC, LG, Motorola, Samsung, Broadcom, Intent, NVIDIA, Qualcomm, and many more.

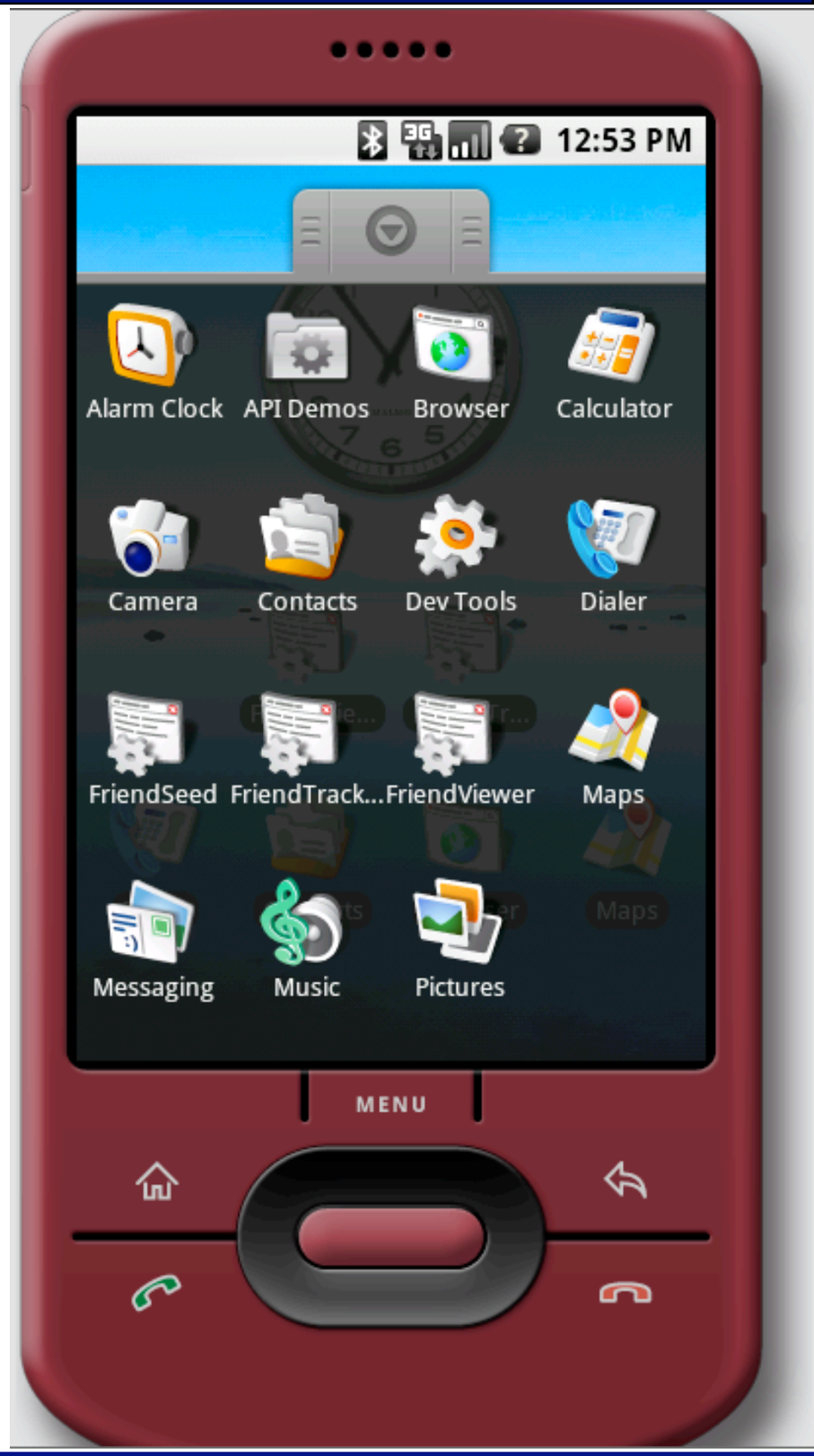
History of Android

- The Open Handset Alliance Vision (from their website)
  - Open interfaces (dialer, SMS, ...)
  - All applications are created equal
  - Breaking down application boundaries
  - Fast & easy application development

- The "Google Phone" rumors go back to at least 2006
  - Google acquired Android, Inc. in July 2005
  - Nov. 2007 - initial SDK release (multiple revs: M3, M5, 0.9, 1.0)
  - Sep. 2008 - T-Mobile announces G1 (available Oct. 2008)
  - Oct. 2008 - Source code released (some Google apps omitted)

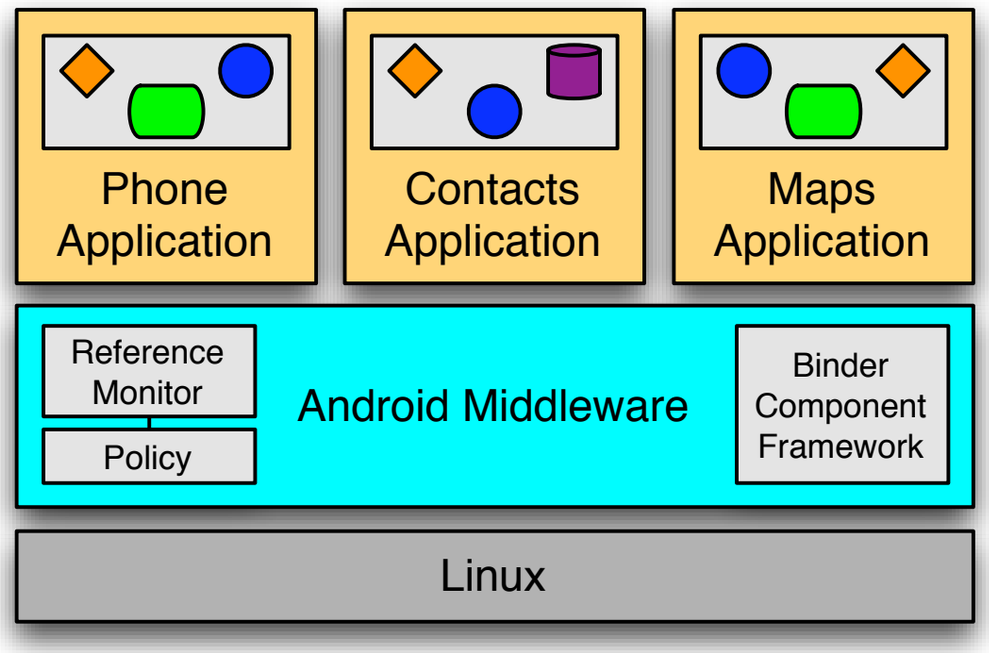
Android Phones

- An Android contains a number of "*applications*"
  - Android comes installed with a number of basic systems tools, e.g., dialer, address book, etc.
  - Developers use the Android API to construct applications.
    - All apps are written in *Java* and executed within a custom Java virtual machine.
  - Each application *package* is contained in a jar file (.apk)
- Applications are *installed* by the user
  - No "app store" required, just build and go.
  - Open access to data and voice services



Architecture

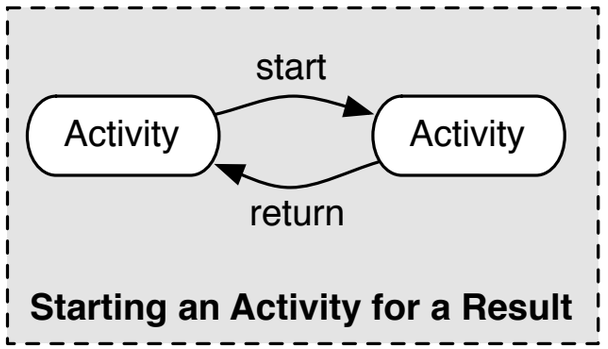
- The Android smartphone operating system is built upon Linux and includes many libraries and a core set of applications.
- The middleware makes it interesting
  - Not focused on UNIX processes
  - Uses the Binder component framework
    - Originally part of BeOS, then enhanced by Palm, now used in Android
  - Applications consist of many *components* of different types
  - Applications interact via components

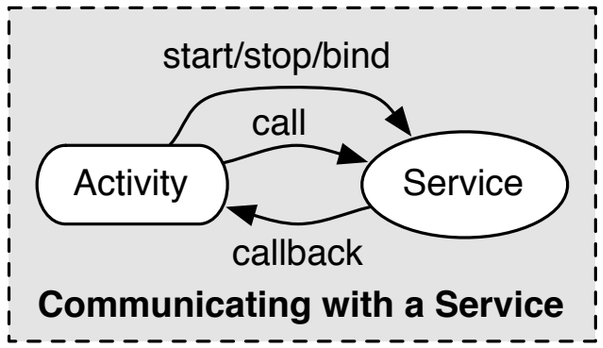


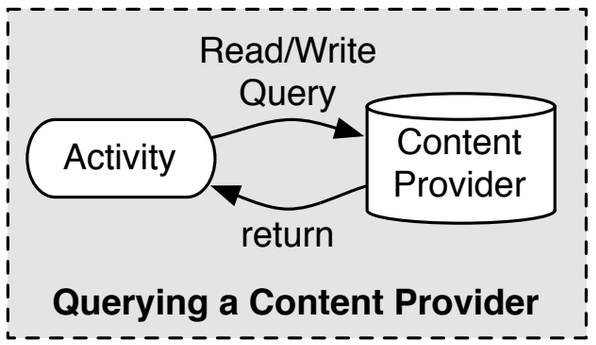
- We focus on security with respect to the component API

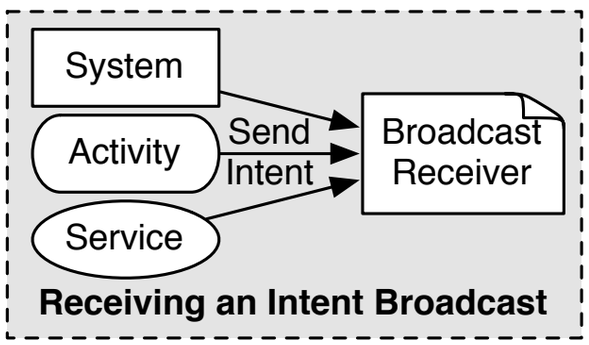
Component Model

- While each application runs as its own UNIX uid, sharing can occur through *application-level* interactions
  - Interactions based on components
  - Different component types
    - Activity
    - Service
    - Content Provider
    - Broadcast Receiver
  - Target component in the same or different application
  - *but first …*









Intents

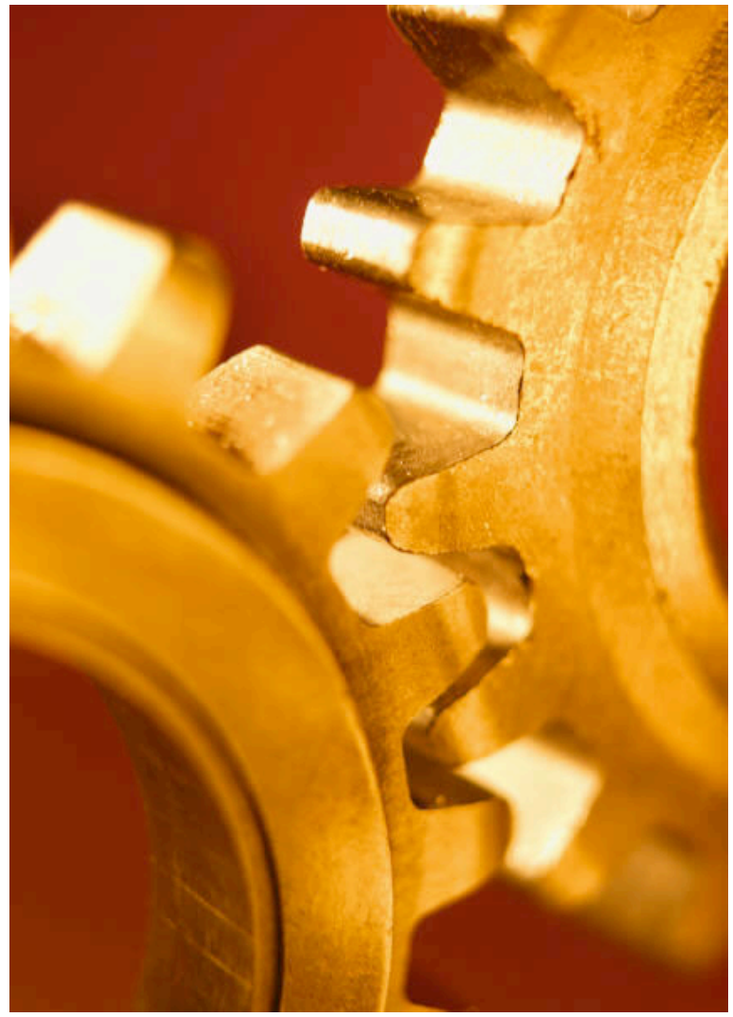
- Intents are objects used as inter-component signaling
  - Starting the user interface for an application
  - Sending a message between components
  - Starting a background service

Activity Component

- The user interface consists of a series of *Activity* components.
- Each Activity is a "screen".
- User actions tell an Activity to *start* another Activity, possibly with the expectation of a *result*.
- The target Activity is not necessarily in the same application.
- Directly or via Intent "action strings".
- Processing stops when another Activity is "on top".

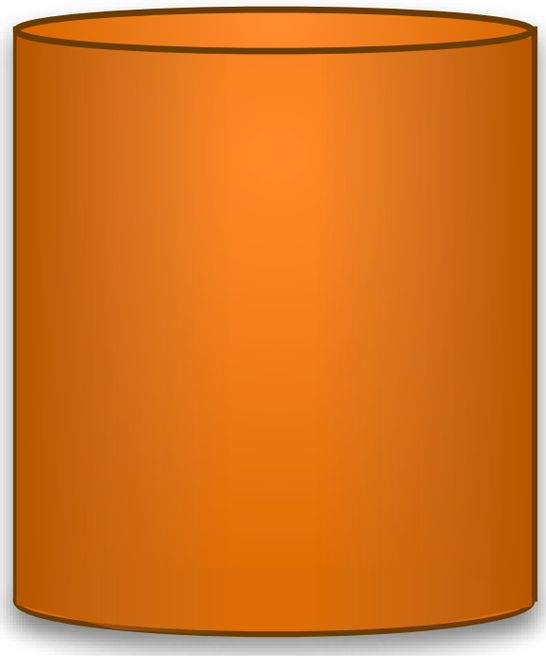
Service Component

- Background processing occurs in *Service* components.
  - Downloading a file, playing music, tracking location, polling, etc.
  - Local vs. Remote Services (process-level distinction)
- Also provides a "service" interface between applications
  - Arbitrary interfaces for data transfer
    - Android Interface Definition Language (AIDL)
  - Register callback methods
  - Core functionality often implemented as Service components
    - e.g., Location API, Alarm service
- Multiple interfaces
  - Control: start, stop
  - Method invocation: bind



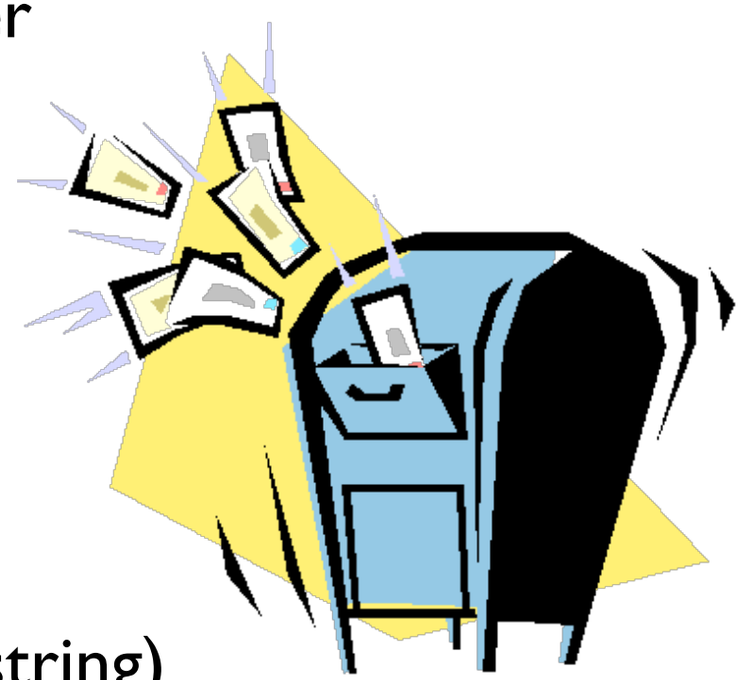
Content Provider Component

- *Content Provider* components provide a standardized interface for sharing data, i.e., content (between applications).
- Models content in a relational DB
  - Users of Content Providers can perform queries equivalent to SELECT, UPDATE, INSERT, DELETE
  - Works well when content is tabular
  - Also works as means of addressing "files"
- URI addressing scheme
  - `content://<authority>/<table>/[<id>]`
  - `content://contacts/people/10`



Broadcast Receiver Component

- *Broadcast Receiver* components act as specialized event *Intent* handlers (also think of as a message mailbox).
- Broadcast Receiver components "*subscribe*" to specific action strings (possibly multiple)
  - action strings are defined by the system or developer
  - component is automatically called by the system
- Recall that Android provides automatic Activity resolution using "action strings".
  - The action string was assigned to an *Intent* object
  - Sender can specify component recipient (no action string)



The Android Manifest

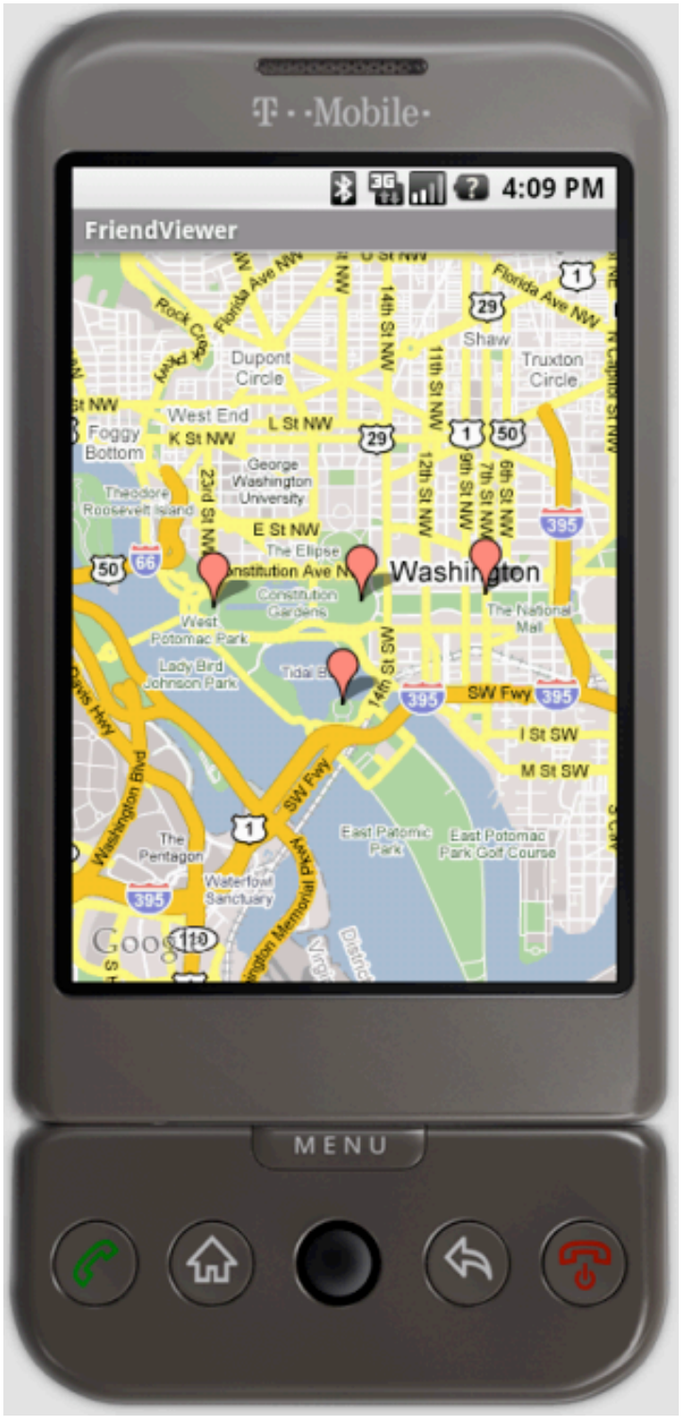
- Manifest files are the technique for describing the contents of an application *package* (i.e., resource file)
- Each Android application has a special `AndroidManifest.xml` file (included in the .apk package)
  - describes the contained components
    - components cannot execute unless they are listed
  - specifies rules for "auto-resolution"
  - specifies access rules
  - describes runtime dependencies
  - optional runtime libraries
  - required system permissions

Manifest Specification

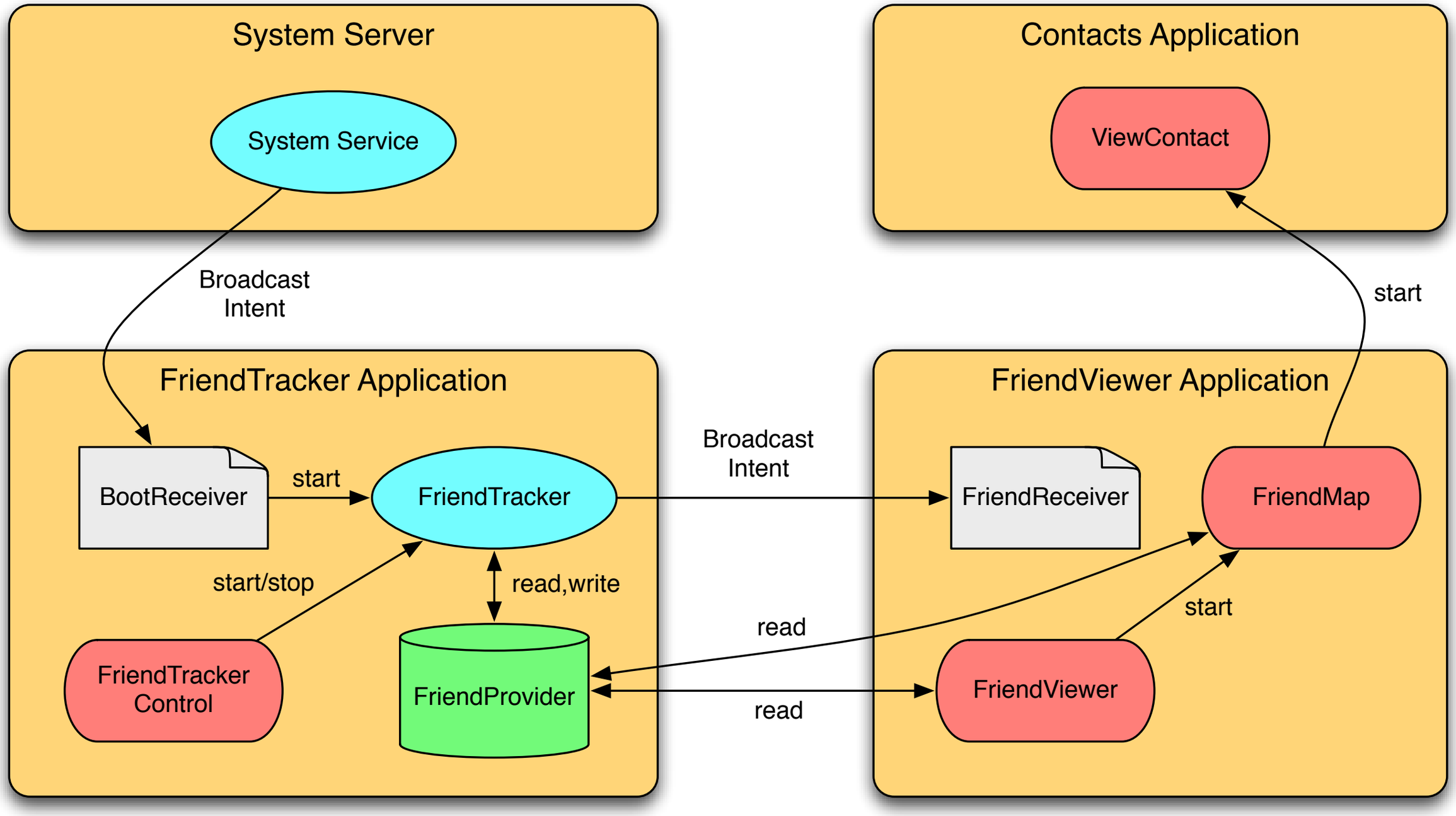
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.siislab.tutorial.friendtracker"
      android:versionCode="1"
      android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".FriendTrackerControl"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:authorities="friends"
              android:name="FriendProvider"
              android:writePermission="org.siislab.tutorial.permission.WRITE_FRIENDS"
              android:readPermission="org.siislab.tutorial.permission.READ_FRIENDS">
        </provider>
        <service android:name="FriendTracker" android:process=":remote"
              android:permission="org.siislab.tutorial.permission.FRIEND_SERVICE">
        </service>
        <receiver android:name="BootReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"></action>
            </intent-filter>
        </receiver>
    </application>

    <!--  Define Permissions -->
    <permission android:name="org.siislab.tutorial.permission.READ_FRIENDS"></permission>
    <permission android:name="org.siislab.tutorial.permission.WRITE_FRIENDS"></permission>
    <permission android:name="org.siislab.tutorial.permission.FRIEND_SERVICE"></permission>

    <!--  Uses Permissions -->
    <uses-permission android:name="org.siislab.tutorial.permission.READ_FRIENDS"></uses-permission>
    <uses-permission android:name="org.siislab.tutorial.permission.WRITE_FRIENDS"></uses-permission>
    <uses-permission android:name="org.siislab.tutorial.permission.FRIEND_SERVICE"></uses-permission>

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-permission>
    <uses-permission android:name="android.permission.READ_CONTACTS"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
</manifest>
```

Example Applications

- FriendTracker Application
  - *FriendTracker* Service to poll for friend locations
    - Broadcasts an Intent when near a friend
  - *FriendProvider* Content Provider to store location of friends
    - Cross references friends with system Contacts Provider
  - *FriendTrackerControl* Activity to start and stop the Service
  - *BootReceiver* Broadcast Receiver to start the service on boot
- FriendViewer Application
  - *FriendViewer* Activity to display list of friend locations
  - *FriendMap* Activity to show friends on a map (on right)
  - *FriendReceiver* Broadcast Receiver to display when near
- Available from http://siis.cse.psu.edu/android_sec_tutorial.html



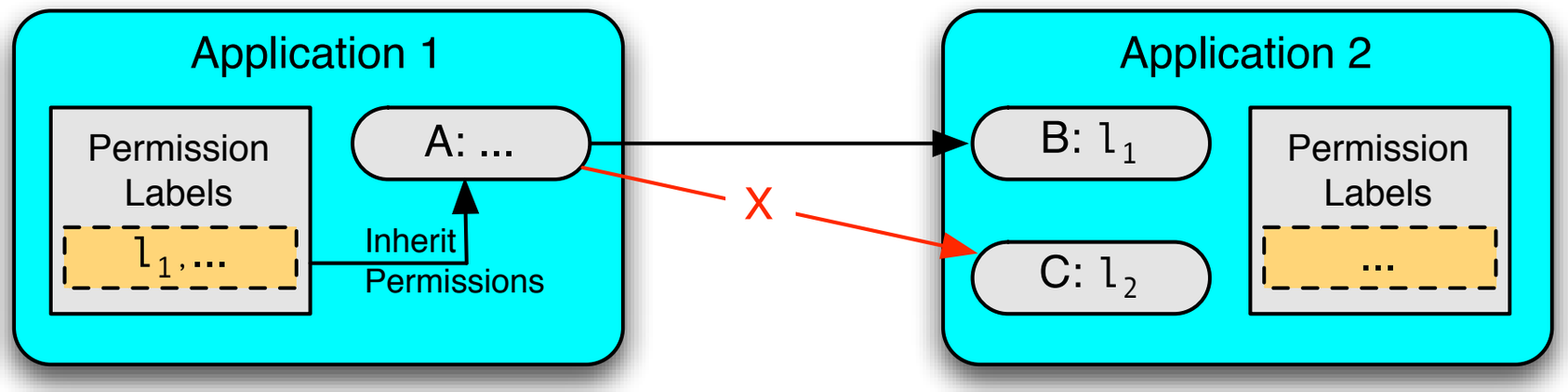
Component Interaction



Defining Security Policy

- Android focuses on *Inter Component Communication* (ICC)
- The Android manifest file allows developers to define an access control policy for access to components
  - Each component can be assigned an access *permission label*
  - Each application requests a list of permission labels (fixed at install)

- Android's security model boils down to the following:



- However, there are a number of exceptions ...

Public and Private Components

- *Exception: Components can be public or private.*
  - Default is dependent on "intent-filter" rules
  - The manifest schema defines an "*exported*" attribute
- *Why:* Protect internal components
  - Especially useful if a "sub-Activity" returns a result
  - e.g., FriendMap Activity in our example

```
<activity android:name="FriendMap" android:exported="false"></activity>
```

- *Implication:* Components may unknowingly be (or become) accessible to other applications.
- *Best Practice:* Always set the "exported" attribute.

Implicitly Open Components

- *Exception*: *If the manifest file does not specify an access permission on a public component, any component in any application can access it.*
- *Why*: Some components should provide "global" access
  - e.g., the main Activity for an Application
  - Permissions are assigned at install-time
- *Implication*: Unprivileged applications have access
  - e.g., FriendReceiver in our example (spoof notification)
- *Best Practice*: Components without access permissions should be exceptional cases, and inputs *must* be scrutinized (consider splitting components).

Implicitly Open Components

- *Excepti* ...ss *permiss* ...ny *applica*
- *Why*: S... ...cess
  - e.g., ...
  - Perm...

```
<activity android:name=".FriendViewer"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>


<receiver android:name="FriendReceiver"
    android:permission="org.siislab.tutorial.permission.BROADCAST_FRIEND_NEAR">
    <intent-filter>
        <action android:name="org.siislab.tutorial.action.FRIEND_NEAR"></action>
    </intent-filter>
</receiver>
```

- *Implication*: Unprivileged applications have access
  - e.g., FriendReceiver in our example (spoof notification)
- *Best Practice*: Components without access permissions should be exceptional cases, and inputs *must* be scrutinized (consider splitting components).

Intent Broadcast Permissions

- *Exception:The code broadcasting an Intent can set an access permission restricting which Broadcast Receivers can access the Intent.*
- *Why*: Define what applications can read broadcasts
  - e.g., the FRIEND_NEAR message in our example
- *Implication*: If no permission label is set on a broadcast, any unprivileged application can read it.
- *Best Practice*: Always specify an access permission on Intent broadcasts (unless explicit destination).

Intent Broadcast Permissions

- *Exception*: *The code broadcasting an Intent can set an access permission restricting which Broadcast Receivers can acc*
- *Why*: D oadcasts
  - e.g., th

```
// Notify any receivers
if (location.distanceTo(floc) <= mDistThreshold) {
    Intent i = new Intent(ACTION_FRIEND_NEAR);
    i.putExtra(FriendContent.Location._ID,
            c.getString(c.getColumnIndex(FriendContent.Location._ID)));
    i.putExtra(FriendContent.Location.NICK,
            c.getString(c.getColumnIndex(FriendContent.Location.NICK)));
    i.putExtra(FriendContent.Location.CONTACTS_ID,
            c.getString(c.getColumnIndex(FriendContent.Location.CONTACTS_ID)));
    sendBroadcast(i, "org.siislab.tutorial.permission.FRIEND_NEAR");
}
```

- *Implication*: If no permission label is set on a broadcast, any unprivileged application can read it.
- *Best Practice*: Always specify an access permission on Intent broadcasts (unless explicit destination).

Pending Intents

- *Exception:* PendingIntent objects allow another application to "finish" an operation for you via RPC.
  - Introduced in the v0.9 SDK release (August 2008)
  - Execution occurs in the originating application's "process" space
- *Why*: Allows external applications to send to private components
  - Used in a number of system APIs (Alarm, Location, Notification)
  - e.g., timer in FriendTracker Service
- *Implication*: The remote application can fill in unspecified values.
  - May influence the destination and/or data integrity
  - Allows a form of delegation
- *Best Practice*: Only use Pending Intents as "delayed callbacks" to private Broadcast Receivers/Activities and always fully specify the Intent destination.

Pending Intents

- *Exceptior* ation to "finish" a
  - Introdu
  - Execut " space
- *Why*: Allc ponents
  - Used i ication)
  - e.g., tin

```
private void scheduleTracking() {
    AlarmManager am = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
    Intent i = new Intent();
    i.setClass(this, FriendTracker.class);
    i.setAction(ACTION_POLL_LOCATIONS);
    PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
    am.set(AlarmManager.ELAPSED_REALTIME_WAKEUP,
            SystemClock.elapsedRealtime() + POLL_INTERVAL, pi);
}

private void cancelTracking() {
    AlarmManager am = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
    Intent i = new Intent();
    i.setClass(this, FriendTracker.class);
    i.setAction(ACTION_POLL_LOCATIONS);
    PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
    am.cancel(pi);
}
```

- *Implication*: The remote application can fill in unspecified values.
  - May influence the destination and/or data integrity
  - Allows a form of delegation
- *Best Practice*: Only use Pending Intents as "delayed callbacks" to private Broadcast Receivers/Activities and always fully specify the Intent destination.

Content Provider Permissions

- *Exception: Content Providers have two additional security features*
  - Separate "read" and "write" access permission labels
  - URI permissions allow record level delegation (added Sep 2008)
- *Why*: Provide control over application data
  - e.g., FriendProvider uses read and write permissions
- *Implication*: Content sharing need not be all or nothing
  - URI permissions allow delegation (must be allowed by Provider)
- *Best Practice*: Always define separate read and write permissions.
  - Allow URI permissions when necessary

Content Provider Permissions

- *Exception: Content Providers have two additional security features*
  - Separate "read" and "write" access permission labels
  - URI permissions allow record level delegation (added Sep 2008)
- *Why*: Prov
  - e.g., Frie

```
<provider android:authorities="friends"
      android:name="FriendProvider"
      android:readPermission="org.siislab.tutorial.permission.READ_FRIENDS"
      android:writePermission="org.siislab.tutorial.permission.WRITE_FRIENDS">
</provider>
```

- *Implication*: Content sharing need not be all or nothing
  - URI permissions allow delegation (must be allowed by Provider)
- *Best Practice*: Always define separate read and write permissions.
  - Allow URI permissions when necessary

Service Hooks

- *Exception: A component (e.g., Service) may arbitrarily invoke the checkPermission() method to enforce ICC.*
- *Why:* Allows Services to differentiate access to specific methods.
  - e.g., .addNick() method of IFriendTracker
- *Implication:* The application developer can add reference monitor hooks
- *Best Practice:* Use checkPermission() to mediate "administrative" operations.
  - Alternatively, create separate Services

Service Hooks

- *Except* … *nvoke* *the che*
- *Why*: A… *cific* metho…
  - e.g.,

```
private final IFriendTracker.Stub mBinder = new IFriendTracker.Stub() {
    public boolean isTracking() {
        return mTracking;
    }

    public boolean addNickname(String nick, int contactId) {
        if (FriendTracker.this.checkCallingPermission(PERMISSION_FRIEND_SERVICE_ADD)
                != PackageManager.PERMISSION_GRANTED) {
            throw new SecurityException("Requires " + PERMISSION_FRIEND_SERVICE_ADD);
        }
        ContentResolver r = FriendTracker.this.getContentResolver();
        ContentValues values = new ContentValues();
        values.put(FriendContent.Location.NICK, nick);
        values.put(FriendContent.Location.CONTACTS_ID, contactId);
        Uri uri = r.insert(FriendContent.Location.CONTENT_URI, values);
        return (uri == null ? false : true);
    }
};
```

- *Implication*: The application developer can add reference monitor hooks
- *Best Practice*: Use checkPermission() to mediate "administrative" operations.
  - Alternatively, create separate Services

Protected APIs

- *Exception: The system uses permission labels to mediate access to certain resource APIs*
- *Why:* The system needs to protect network and hardware resources
  - e.g., Applications request the `android.permission.INTERNET` label to make network connections.

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

- *Implication:* Allows the system or a user to assess how "dangerous" an application may be.
- *Best Practices:* Judiciously request permissions for protected APIs

Permission Protection Levels

- *Exception: Permission requests are not always granted*
  - Permissions can be:
    - normal - always granted
    - dangerous - requires user approval
    - signature - matching signature key
    - signature or system - same as signature, but also system apps
- *Why*: Malicious applications may request harmful permissions
  - e.g., privacy implications of receiving FRIEND_NEAR
- *Implication*: Users may not understand implications when explicitly granting permissions.
- *Best Practice*: Use signature permissions for application "suites" and dangerous permissions otherwise
  - Include informative descriptions

Permission Protection Levels

- *Exception*: *Permission requests are not always granted*
  - Permissi
    - normal
    - dangerc
    - signatur
    - signatur

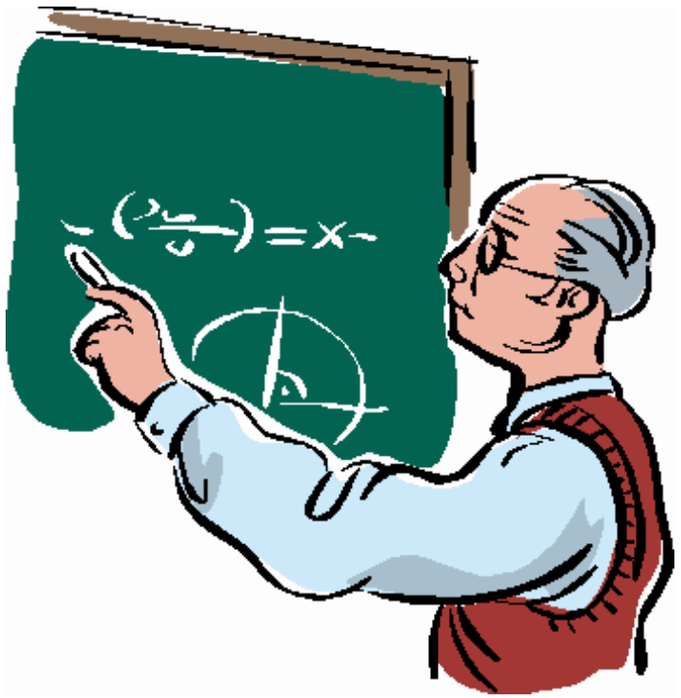
```
<permission android:name="org.siislab.tutorial.permission.FRIEND_NEAR"
    android:label="@string/permlab_friendNear"
    android:description="@string/permdesc_friendNear"
    android:protectionLevel="dangerous">
</permission>


<string name="permlab_friendNear">receive friend near notification</string>
<string name="permdesc_friendNear">Allows an application to receive a
    notification when a friend is near. Malicious applications may
    monitor your proximity to your friends</string>
```

s

- *Why*: Malicious applications may request harmful permissions
  - e.g., privacy implications of receiving FRIEND_NEAR
- *Implication*: Users may not understand implications when explicitly granting permissions.
- *Best Practice*: Use signature permissions for application "suites" and dangerous permissions otherwise
  - Include informative descriptions

Lessons in Defining Policy

- Relatively straightforward model with policy defined in the manifest file ... but many exceptions
- Some thought is needed to avoid ...
  - "Spoofing" Intent messages (FriendReceiver)
  - Privacy leaks (e.g., FRIEND_NEAR broadcast)
- The policy expands into the code
  - Broadcast permissions, checkPermission(), etc
- Keeping malicious applications from acquiring permissions is tricky



Install-time Verification

- *Deficiency*: Android does not have a way to holistically evaluate system and application policy or specify security goals
  - For example, to evaluate if the system and installed applications fulfill some security requirement
  - *Will granting a permission break the phone's security?*

- Kirin - enhanced installer we have been developing
  - Extracts policy from the manifest files of all applications
  - Uses Prolog to generate automated proofs of compliance of provided "policy invariants"
  - Evaluation must only be performed at install-time, and therefore does not impact runtime performance

Summary

- It is highly likely that Android is going to be installed many millions of cell phones within the next 12 months.
  - If you are building applications, you need to be aware of what access other applications are going to be able to do to your user ...
  - *Take away*: *be defensive*!
- Android security is complex beast, and this tutorial is a first step towards understanding and dealing with it.
- If you want to learn more, consult Android documentation (the security model description is lacking),
- Recommendation: http://code.google.com/android/intro/

Thank you

Presentation Slides and Code Examples
http://siis.cse.psu.edu/android_sec_tutorial.html

Systems and Internet Infrastructure Security Lab (SIIS)
Department of Computer Science and Engineering
The Pennsylvania State University
http://siis.cse.psu.edu

Google Android SDK and Documentation
http://code.google.com/android

Android Platform Source Code
http://source.android.com