A qbit on Android Security

Sergey Gorbunov

One of the problems with modern desktop operating system is that there is no isolation between applications and the system. As we saw in class, a buggy application can potentially allow an attacker to gain full access to the system. Therefore, we *have to trust* all applications that we install and use on daily basis. To overcome these limitations, Google takes a different approach in application security. On the very high level, it isolates applications in separate virtual machines. By default, applications cannot freely communicate with other applications and services. Hence, compromising one application, cannot not allow the attacker to take over the system. Yet, the Android security model is complicated, as we discuss below, and bad coding practices can lead to serious security vulnerabilities and data leakage.

1 Basics of Android Application

Each Android application is isolated in a separate Davlik VM running on Linux Kernel (See Figure 1). It gets assigned a separate UID/PID in the system. Assigning separate UID for each application gives extra flexibility in managing inputs/outputs of the application. Application do not have a main() entry point. Instead, they are divided into separate components. Interaction between components is controlled via permissions labels. Let's take a look at a simple example in Figure 2 developed for demo purposes of Android Security by Enck, Ongtang and McDaniel [1].

There are two applications: FriendTracker and FriendViewer, each divided into separate components.

- **Services**: FriendTracker is a service that can constantly run at the background. It monitors the position of certain friends.
- **Activities**: FriendTrackerControl, FriendViewer and FriendTracker: these are typically used to interact with users. For example, FriendViewer is used to list all friends and their positions on the phone. Only one activity can have the token for the screen and the keyboard at a time.
- **Content Providers**: FriendProvider is used to store and share information through a relational database. Each providers describes an *authority* which other components can use to perform "SQL like" queries.
- **Broadcast Receivers**: These are used to listen for incoming calls to a specific destination and react appropriately.

```
public class FriendReceiver extends BroadcastReceiver {
       private static final String ACTION_FRIEND_NEAR =
           "org.siislab.tutorial.action.FRIEND_NEAR";

       /* (non-Javadoc)
```

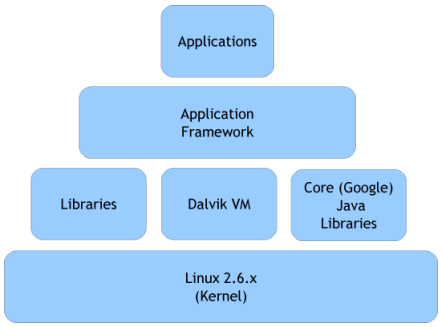

Figure 1: Basic Android Framework

```
         * @see android.content.BroadcastReceiver#onReceive(android.content.Context,
             android.content.Intent)
         */
        @Override
        public void onReceive(Context context, Intent intent) {

               if (ACTION_FRIEND_NEAR.equals(intent.getAction())) {
                      Bundle extras = intent.getExtras();
                      if (extras != null) {
                             String nick =
                                 extras.getString(FriendContent.Location.NICK);
                             if (nick != null) {
                                    Toast.makeText(context, "Near " + nick,
                                        Toast.LENGTH_SHORT).show();
                             }
                      }
               }

        }
}
```

The developer specifies the components in the manifest.xml file. The specific activity launched at the start time is marked using the metadata.

```
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER"/>
```

2 Component Interaction

Each application runs with a separate UID, allowing to minimize post-attack damage. Interaction between components is performed via /dev/binder (Figure 3). This file is world readable and world writable. The decisions about granting input/output access between two components is performed via labels. That is, the developer assigns labels to the application and its components. The ICC reference monitor is responsible for testing whether the source application has the necessary labels.

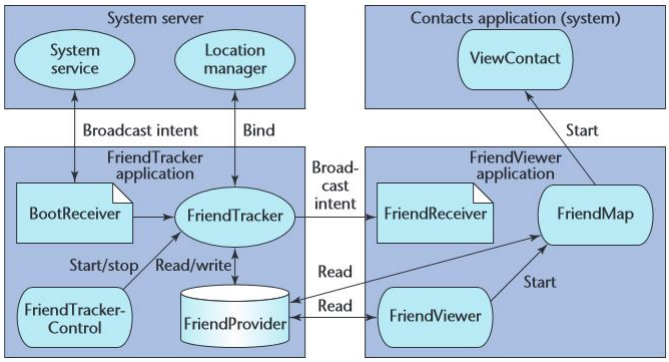

Figure 2: FriendTracker and FriendViewer Example Applications

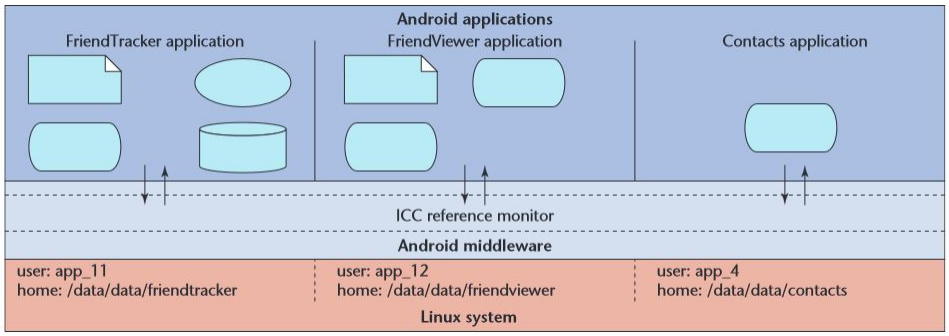

Figure 3: Interaction Between Application Components

The basic way of interaction is by creating an intent object, which specifies the *address* or an *action string* (which allows the system to choose the addressee. For example, "VIEW" action string will direct to the preferred image viewer).

The application must define a list of permissions that it wants to request upon installation. These cannot be changed unless the application is reinstalled. All application components inherit these permissions.

```
<uses-permission android:name="org.siislab.tutorial.permission.FRIEND_SERVICE"/>
<uses-permission android:name="org.siislab.tutorial.permission.FRIEND_NEAR"/>
<uses-permission android:name="org.siislab.tutorial.permission.BROADCAST_FRIEND_NEAR"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

In addition, each component must declare the permission that will be used to control access to it. All this is done in the metafile.

```
<service android:name="FriendTracker"
    android:permission="org.siislab.tutorial.permission.FRIEND_SERVICE"
```

3 Security Considerations

- **Private Components** It is possible to set the component to be private by declaring *exported* attribute to false. In this case, no external application will be able to access it.
- **Implicitly Open Components** In some cases, it is a-priori unknown which permission is required to access an application (for example, image viewer installed on the system). In this case, the user can leave the permissions set empty which would allow any application to open it.
- **Broadcast Intents** These could potentially leak some information to unauthorized applications. For example:

```
// 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);
```

  To fix this, we could specify an addition label upon sending which the receiving applications must have.

```
sendBroadcast(i, "org.siislab.tutorial.permission.FRIEND_NEAR") ;
```

- **Content Providers Security** To allow more fine-grained access to the content providers, the developer can specify separate read and write permissions for the component.
- **Functional Security Checks** The developer can choose to use checkPermissions() method to allow/deny access to a certain function at the run-time of the program. This complicates android security even further, since to understand the security of an application, we need to check its code and not just the metadata files.
- **Pending Intents** Android allows to "outsource" filling some parts of an intent to a different component (similar to prepared statements, where variables are filled up by the user). However, this means that the component that gets to complete the intent has the permission level of the outsourcing component. Hence, the outsourcing component must trust it.

To summarize, Android development framework allows great flexibility in designing secure applications: the programmers can isolate application and its components from the rest of the system, trying to control the information flow. However, as we saw above, this flexibility comes at the cost. Developers are granted with a variety of options and methods for enforcing the permission checks. Hence, defining, implementing and understanding the permissions of applications is not a trivial tasks.

References

[1] W. Enck, M. Ongtang, and P. McDaniel. Understanding *Android Security*. IEEE Security & Privacy Magazine, 7(1):5057, January/February 2009. `http://siis.cse.psu.edu/android_sec_tutorial.html`