Lab3: Android

Noak Jönsson, Felipe San Martín

May 2024

1 Intent Sniffing

1.1 Android Broadcast System

Android allows apps to both send and receive broadcast messages from or to other apps, as well as the Android system itself. These broadcast messages follow the publish-subscribe design pattern, meaning that messages sent by a publisher are not targeted to a specific predefined receiver, rather they are sent to all services which subscribe to messages of a certain class.

This model is beneficial as it allows for loose coupling between apps, and also between the system and all possible apps installed on the system, leading to better scalability.

Examples of system broadcast messages include, notifying a successful boot of the system, notifying changed status of airplane mode, and notifying disconnection of a power supply.

1.2 Implementation

Both apps use context registered receivers for inter- and intra-app messaging. By registering a Receiver object with a filter for the same intents used by LocationApp and WeatherApp our service can capture all broadcasts and extract the location data as these broadcasts are sent system wide. When receivers are registered, the android system is informed that the app containing the receiver should be passed all messages which matches the intent they have registered for. Unless the app which sends the broadcast has explicitly stated which apps can receive broadcasts, or what permissions are required to receive the broadcasts, any malicious or benign application can access the message sent provided they are registered to receive messages of that intent.

```
SnifferBroadcastReceiver wReceiver ;
SnifferBroadcastReceiver lReceiver ;

static final String WEATHER_RECEIVER = "tcs.lbs.weather_app.WeatherBroadcastReceiver";
static final String LOCATION_RECEIVER = "tcs.lbs.locationapp.MainActivityReceiver";

protected void onCreate(Bundle savedInstanceState) {
    IntentFilter wFilter = new IntentFilter();
    wFilter.addAction(WEATHER_RECEIVER);
    wReceiver = new SnifferBroadcastReceiver();
    registerReceiver(wReceiver, wFilter);

    IntentFilter lFilter = new IntentFilter();
    lFilter.addAction(LOCATION_RECEIVER);
    lReceiver = new SnifferBroadcastReceiver();
    registerReceiver(lReceiver, lFilter);
}

protected class SnifferBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        location = intent.getParcelableExtra("Location");
        if (location!=null) {
            double latitude = location.getLatitude();
            double longitude = location.getLongitude();
```

```
                LatitudeTextView.setText(String.valueOf(latitude));
                LongitudeTextView.setText(String.valueOf(longitude));
            }
        }
    }
```

1.3 Intra-App Broadcast Sniffing

In order to preserve the publisher-subscriber design pattern while not leaking data from intra-app broadcasts LocationApp could utilize explicit broadcasts. This could be done by using either the Intent setComponent(ReceiverFoo) method, which would send the broadcast method to only receivers of class ReceiverFoo, or by using the Intent setPackage(foo), which would make the broadcast visible to all receivers registered in the application foo.

```
locationAppIntent.setPackage("tcs.lbs.locationapp");
sendBroadcast(locationAppIntent);
```

There is a now deprecated android class LocalBroadcastManager, which previously could be used to restrict broadcast to within an application context. This would allow intra-app message passing when using sendBroadcast, without the possibility of malicious apps intercepting the data. Since deprecation android recommends using a LiveData class to facilitate intra-app messaging, although it adheres to an Observable design pattern, rather than publisher-subscriber.

1.4 Inter-App Broadcast Sniffing

Inter-app broadcast sniffing could be managed by using predefined permissions when sending broadcasts as well as registering receivers. These permissions would be defined by the broadcasting app, and all receivers would have to provide these permissions in their manifest.

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

By restricting the broadcast to apps that have already been granted permission to access the device location, the location data will not leak to malicious apps, as these permissions must be granted by the user at runtime.

The LocationApp could also utilize the Intent setPackage(weather_app) method to specify which applications can receiver broadcast messages. This would result in explicit broadcasts which could be harder to scale

2 Confused Deputy Attack

2.1 DataBaseActivity functionality

Database activity is in charge of managing the use of the application, specifically what is related to the database, such as creating, deleting or displaying a note. It receives an ”intent” sent from MainActivity, which indicates what should be done with the database. This is done thanks to different helpers, such as dataBaseHelper. DataBaseHelper is a class that extends SQLite, and uses it to modify database content.

2.2 Root issue

The root of the problem is that DataBaseActivity does not check that the intent comes from an app with privileges given by the user, so it simply executes with respect to the intent received with its own permissions.

2.3 Solutions for the issue

1. The first way to solve this problem is to check in DataBaseActivity that the intents come from the same app for which they were created, thus preventing misuse by other applications.
2. The second option is to explicitly ask for a specific permission to the user, required each time DataBaseActivity is used. This can be a custom permission defined by the Notes app, which would then need to be requested by, and granted to, any other app wanting to use DataBaseActivity. This way only authorized applications would be able to modify the database given user authorization.

2.4 Pros and Cons of our solutions

2.4.1 Check intents in DataBaseActivity

- Pros: Increases security without being invasive to the user, since the user does not perceive the implementation directly. This solution is possible because for this context no other application should be able to modify the database of the notes application.
- Cons: Adds a possible limitation to the use of the application, if for example the application needs to communicate with other legitimate applications, this additional verification could hinder or prevent the interaction.

2.4.2 Use of user permissions

- Pros: By implementing explicit authorization, the user is given the opportunity to deny access to sensitive data, which increases transparency and control over personal information.
- Cons: This generates a less fluid user experience, as it can be annoying for users to have to repeatedly grant permissions to the application, decreasing the usability of the application.

2.5 Implementation of solution 1

We implemented in DataBaseActivity a check (before modifying the database given an intent), if the intent received comes from the same application, the code is as follows:

```
// Check if the Intent sender is from the same app
if (!isIntentFromSameApp()) {
    // If not, finish the activity
    finish();
    return;
}

// Function used to check if Intent sender is from the same app
private boolean isIntentFromSameApp() {
    // Get the package name of the Intent sender
    String senderPackageName = getCallingPackage();

    // Get the package name of our app
    String appPackageName = getApplicationContext().getPackageName();

    // Compare the package names
    return appPackageName.equals(senderPackageName);
}
```

With this code, we check after the intent is received, but before the action specified is executed, if the intent comes from the same app, as follows:

```
...
case ACTION_DeleteItem:
    Intent deleteIntent = getIntent();
    String deleteID = deleteIntent.getStringExtra(NOTE_ID);

    // Check if the Intent sender is from the same app
    if (!isIntentFromSameApp()) {
        // If not, finish the activity
        finish();
        return;
    }

    if (!deleteID.equals(NEW_ID))
    {
        dataBaseHelper.remove(deleteID);
    }
```

```
        setResult(RESULT_OK);
        finish();
        break;
    ...
```

3 Contribution of each member

3.1 Intent Sniffing

- Noak: Research, implementation of Intent Sniffer and mitigations. Wrote associated report
- Felipe: Research, code study and implementation of the Intent Sniffer.

3.2 Confused Deputy Attack

- Noak: Research on topic, study of the code.
- Felipe: Research on the topic, study of the code and implementation of the Confused Deputy Attack and the solution. Wrote the associated report.