Detecting Passive Content Leaks and Pollution in Android Applications

Yajin Zhou and Xuxian Jiang
North Carolina State University
Apps Are Becoming Popular

- Initial release: 10/2008
- 675,000 apps
- 25 billion downloads
- 09/2012
- 800,000 apps
- ~30 billion downloads
- 02/2013

- Google Play Logo
- Graph showing growth of apps and downloads.
Apps Are Managing User Data

- Messages
- Friends
- Browser Histories
- Bank Accounts
Content Providers

- Manage access to a structured set of data

- **By default** are *open* to all apps on the phone (before Android 4.2)

Any potential security risks?
A Motivating Example

- GO FBWidget: popular Android app with more than 1 million installs
A Motivating Example

```java
final class h implements Facebook.DialogListener {
    public void onComplete(Bundle paramBundle) {
        String token = FaceBookChooserActivity.a(this.a).getAccessToken();
        ContentValues c = new ContentValues();
        c.put("accesstoken", token);
        ContentResolver resolver = this.a.getApplicationContext().getContentResolver();
        resolver.insert(FacebookProvider.SETTING_CONTENT_URI, c);
    }
}

class FacebookProvider implements extends ContentProvider {
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        SQLiteDatabase db = this.aq.getWritableDatabase();
        SQLiteQueryBuilder q = new SQLiteQueryBuilder();
        q.setTables("settings");
        Cursor c = q.query(db, projection, selection, selectionArgs, null, null, sortOrder);
        return c;
    }
}
```

get Facebook access token
insert access token into internal database
content provider implementation

public read interface of content providers
API that actually queries internal database
A Motivating Example

- Can be exploited to leak private data
  - *Access token*, Facebook posts

Automatically log into user’s Facebook account and make posts
Our Work

- Systematically study two vulnerabilities: content leaks and content pollution
  - 2.0% and 1.4% of apps are susceptible, respectively
  - Types of information leaked
    - SMS messages, contacts, user credentials, ...
  - Possible side-effects
    - Block SMS messages and phone calls
    - Download apps and prompt for installation
System Design

Android apps

App Repository

Select Candidate Apps

Determine Vulnerable Apps

Classify Vulnerable Apps

Confirm Vulnerabilities

Generate Inputs

Find Execution Paths

ContentScope

Report
Find Execution Paths

- From public interfaces of content providers to functions that actually operate on internal database
Find Execution Paths

- Function call graph
  - Object reference resolution
  - Call graph discontinuity

```
Thread.start()
```

```
Thread.run()
```

```
Handler.sendMessage()
```

```
Handler.handleMessage()
```

```
AsyncTask.execute()
```

```
AsyncTask.onPreExecute()
```

```
AsyncTask doInBackground()
```

```
AsyncTask.onPostExecute()
```
Generate Inputs

- Generate control flow graph
- Obtain constraints
- Resolve constraints
Generate Inputs

- Android specific APIs
  - UriMatcher

If (sUriMatcher.match(uri))

`sUriMatcher.addURI( "com.example.app.provider", "example_table", 1);`

uri

What’re satisfied inputs?

!= 1

exception

== 1

Call internalQuery()

content://com.example.app.provider/example_table
Confirm Vulnerabilities

- Feed generated inputs into a test app
- Invoke public interfaces of content providers
  - query(), insert(), ...
- Determine the existence of vulnerabilities based on return value
  - query(): Cursor object
  - insert(): URI object
System Implementation

- Around 6,500 SLOCs
- Public interfaces of content providers
  - query(), openFile()
  - insert(), update()
- APIs that actually read or write internal database
  - SQLiteDatabase.query(), SQLiteDatabase.insert(),
    SQLiteQueryBuilder.query(), ...
Evaluation

- Dataset: 62,519 free apps
  - Sources: Google Play and ten other Android markets
- Time: February 2012

![Pie chart showing distribution between Google Play and Other Markets]

- Google Play: 35,047 (56.06%)
- Other Markets: 27,472 (43.94%)
Overall Results

- **Content Leaks**
  - 1,279 (2.0%)
  - 844
  - 435

- **Content Pollution**
  - 871 (1.4%)
  - 473
  - 398

- **Both**
  - 694 (1.1%)
  - 460
  - 234
# Main Types of Leaked Data

<table>
<thead>
<tr>
<th>Category</th>
<th># of apps</th>
<th>Representative App</th>
<th># of Installs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMS messages</td>
<td>268</td>
<td>Pansi SMS</td>
<td>500,000 – 1,000,000</td>
</tr>
<tr>
<td>Contacts</td>
<td>128</td>
<td>mOffice – Outlook sync</td>
<td>100,000 – 500,000</td>
</tr>
<tr>
<td>Private information in IM Apps</td>
<td>121</td>
<td>Messenger With You</td>
<td>10,000,000 – 50,000,000</td>
</tr>
<tr>
<td>User credentials</td>
<td>80</td>
<td>GO FB Widget</td>
<td>1,000,000 – 5,000,000</td>
</tr>
<tr>
<td>Browser History</td>
<td>70</td>
<td>Dolphin Browser HD</td>
<td>10,000,000 – 50,000,000</td>
</tr>
<tr>
<td>Call logs</td>
<td>61</td>
<td>Droid Call Filter</td>
<td>100,000 – 500,000</td>
</tr>
<tr>
<td>Private information In social network apps</td>
<td>27</td>
<td>Sina Weibo</td>
<td>100,000 – 500,000</td>
</tr>
</tbody>
</table>
Side-effects of Content Pollution

- Block SMS messages and phone calls: by manipulating security settings
  - DW Contacts

- Download apps and prompt for installation
  - Baidu Appsearch, Qihoo Browser
Vulnerable Security Apps

- Mobile Security Personal Ed.
  - Leak browser histories
- QQPimSecure, Anguanjia
  - Leak SMS, phone call logs
  - Block SMS and phone calls
Possible Mitigations

- App Developers
  - Patch their vulnerable apps
- Platform provider (Google)
  - Change the default setting of content provider interface
Possible Mitigations

- By Google: content providers are no longer exported by default on Android since 4.2
  - Developers need to **explicitly** change manifest file
    - Set targetSdkVersion to 17 (or higher)
  - Problems remain on old Android versions
    - The API level of **98.6%** Android devices are less than 17 on February 04, 2013 [1]

Possible Mitigations

- By Google, content providers are no longer exported by default since Android 4.2.
  - Developers need to explicitly change the manifest file.
    - Set `targetSdkVersion` to 17 (or higher).

- Problems remain on old Android versions.
  - The API level of 98.6% of Android devices are less than 17 on February 04, 2013.

Related Work

- Smartphone privacy
  - TaintDroid [Enck et al., OSDI 10], AdRisk [Grace et al., ACM WiSec 12] ...

- Confused deputy
  - Woodpecker [Grace et al., NDSS 12], Permission Re-Delegation [Felt et al., USENIX Security 11] ...

- Vulnerability detection
  - BitBlaze [Song et al., ICISS 08], KLEE [Cadar et al., USENIX Security 08] ...
Conclusion

- Systematically study two vulnerabilities: content leaks and content pollution
  - 2.0% and 1.4% of apps are susceptible, respectively
- Types of information leaked
  - SMS messages, contacts, user credentials, ...
- Possible side-effects:
  - Block SMS messages and phone calls, ...
Q&A

Yajin Zhou
http://yajin.org
(yajin_zhou@ncsu.edu)