ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications

Jinyung Kim, Yongho Yoon, and Kwangkeun Yi
Programming Research Laboratory
Seoul National University
Seoul, Korea
{jykim,yhyoon,kwang}@ropas.snu.ac.kr

Junbum Shin
SW R&D Center
Samsung Electronics
Suwon, Korea
junbum.shin@samsung.com

A. Problem

Smartphone applications can steal users' private data and send it out behind their back. Smartphones store various personal data, such as phone identifiers, location information, and contacts. Third-party applications, which can be downloaded freely at markets, frequently access the data. Most of the applications do so to explore the fun and utility of smartphone technology. However, such accesses also raise concerns and issues of privacy risk.

Android's permission-based approach is not enough to ensure the security of private information. Android requires application developers to declare the permissions so their applications can access users' private information. However, the permissions does not let you know the actual trace of private data. It is uncertain if an application only accesses private data locally, or sends the data out. Also, developers tend to request more permissions than what they need. As a result, users also tend to care less about the permissions when they install applications.

B. Our solution

We developed a static analyzer ScanDal that detects privacy leaks in Android applications. ScanDal determines if there exists any flow of data from an information source through a sink. ScanDal is a sound analyzer. It covers all possible states which may occur when using the application. In other words, ScanDal can detect every possible privacy leak in the application.

We analyzed 90 popular applications using ScanDal from Android Market and detected privacy leaks in 11 applications. We also analyzed 8 known malicious applications from third-party markets and detected privacy leaks in all 8 applications.

C. Example

The following is a simple example of an interprocedural privacy leak ScanDal detected. The example code is from the Dalvik bytecode of *Google Wallpaper 4.2.2*. This application sends a device ID, called IMEI, to the content server. At line 2, the application gets device ID by calling the `getDeviceId` API and stores it in a global variable. After that, in the `getLocale_version_IMEI_W_H` method, the IMEI is loaded and is appended to some other string values and returned. The returned string is passed to the `getSearchURL` method, and also manipulated and returned to `initTagWebView`. Finally, the string that contains IMEI is made into a URL and sent to the content server of the application.

```
 1 Wallpapers.onCreate()
 2     callv TelephonyManager.getDeviceId()
 3     move-result r3
 4     puts r3 eWallpaperConst.IMEI
 5
 6 XMLTools.getLocale_version_IMEI_W_H()
 7     gets r5 eWallpaperConst.IMEI
 8     callv StringBuilder.append(r4,r5)
 9     move-result r4
10     callv StringBuilder.toString(r4)
11     move-result r4
12     return r4
13
14 XMLTools.getSearchURL()
15     calld getLocale_version_IMEI_W_H
16     move-result r2
17     callv StringBuilder.append(r1,r2)
18     move-result r1
19     callv StringBuilder.toString(r1)
20     move-result r0
21     return r0
22
23 SearchTagsActivity.initTagWebView()
24     calld XMLTools.getSearchURL(r1)
25     move-result r1
26     callv WebView.loadUrl(r1)
```