P-Lint: A Permission Smell Detector for Android Applications

Colton Dennis, Daniel E. Krutz and Mohamed Wiem Mkaouer
Department of Software Engineering
Rochester Institute of Technology
Rochester, NY, USA
Email: {crd6283, dxkvse, mwmvse}@rit.edu

***Abstract*—Android is built upon a permission-based structure, where apps require access to specific permissions in order to carry out specific functionalities. While Android has provided a set of best practices intended to aid the developer in properly defining and manipulating these permissions on their source code, developers do not always adhere to these guidelines. Although some of the resulting issues may be minor and lead to slight user confusion, other mistakes may create more serious privacy and security related issues. We've defined improper usage of these permission best practices to be *permission smells* to indicate possible permissions related syntactic issues and have created a tool *P-Lint* to assist in the identification of these smells on the source code. P-Lint's goal is to not only help developers create better, more secure apps by providing guidance on properly using permissions, but also in allowing researchers to better understand the common permission smells through empirical analysis on existing apps. P-Lint is publicly available on the project website: https://p-lint.github.io**

I. Introduction

Beginning with Android Marshmallow (API 23), developers may now ask users to request permissions at run-time, and users may choose to grant only some of the app's requested permissions. This significantly differs from previous versions of Android where developers would only be allowed to ask for permissions upon the installation of the app, and users would have to accept the permissions in an all-or-nothing fashion. Android introduced several permission 'best practices' for using this new permissions model. Not adhering to some of these rules can have more profound effects in comparison with other rules.

A 'code smell' is a symptom of a bad programming practice, and not a syntactic error, i. e., not that an issue necessarily exists. Similarly, we define *permission smells*, as an indication of a permission-related bad programming practice. Some examples of permission smells include not adhering to Google's permission best practices guideline and misusing the *checkSelfPermission()*, requesting permissions when Intents are advised to be used, or possible misuse of custom permissions. We have designed and implemented P-Lint[1] (Permission-Lint) to detect permission smells. The primary contributions of P-Lint are:

- Assist developers with developing better permission-related statements and methods, and help them adhere to defined standards.
- Provide a tool for researchers to analyze a large number of existing apps for permissions related bad practices, which may have led to potential privacy or security vulnerabilities.

To our knowledge, this is the first tool that analyzes Android apps for proper permissions usage from a standards perspective. P-Lint differs from tools such as PScout [2] and Stowaway [3] since it does not focus on merely identifying the permission-gap, but in performing permissions checks based on best practices.

II. Tool Overview

We designed P-Lint to be both easy to adopt and use. P-Lint begins by reverse engineering the Android apk file using existing tools including Apktool[2], dex2jar[3], and JD-Core-java[4]. An overview of P-Lint is shown in Figure 1.

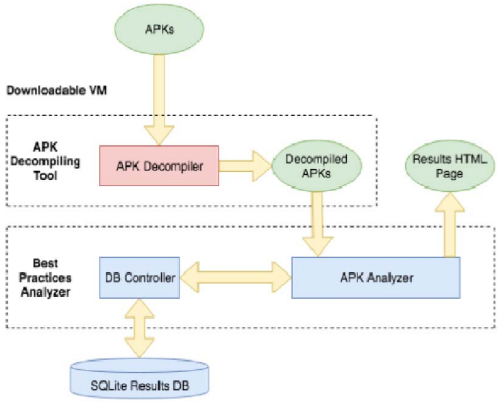


Fig. 1. P-Lint Overview

Using static analysis, P-Lint examines the reverse engineered code for the defined best practices use cases. Since

[1]https://p-lint.github.io

[2]https://ibotpeaches.github.io/Apktool/
[3]https://sourceforge.net/projects/dex2jar/
[4]https://github.com/nviennot/jd-core-java

978-1-5386-2669-6/17 $31.00 © 2017 IEEE
DOI 10.1109/MOBILESoft.2017.24

changes to the Android permission model and best practices guidelines are a certainty, we created P-Lint to be as easily extensible as possible and will maintain the tool to account for changes in Android.

P-Lint's results are stored in an html file to make the results easy to interpret, and are also stored in a database to make the results easy for researchers to query. Some of the permission smells detected by P-Lint include:

1) Improper use of custom permissions
2) Requesting permissions when Intents could be used
3) Improper usage of functionality such as *checkSelfPermission()* and *shouldShowPermissionRationale()*
4) Inconsistent or confusing rationales

To illustrate an example of these bad permission practices, we consider the following example:

Missing checkSelfPermission(): Prior to executing code that requires a particular permission, the method *checkSelfPermission()* needs to be called to check if the user has granted permission for the app to access the Android functionality that falls under this permission. If the user has not as yet approved, denied this permission request on a prior occasion or even reverted the approved permission there is a high chance of the app crashing or not performing it's intended functionality. *requestPermissions()* can be called if *checkSelfPermission()* returns false.

```
// Request permission access every time
void someMethod(){
  ActivityCompat.requestPermissions(this, new String
      [ {Manifest.permission.READ_CALENDAR},
      PERMISSION_READ_CALENDAR);
}
```

Listing 1. Failure to Use CheckSelfPermission

For instance, the following code snippet in Listing 1 demonstrates the usage of PERMISSION_READ_CALENDAR inside a given function with a developer's assumption that the permission has been eventually granted. This assumption may be incorrect if the user does not grant the app to access that permission during its runtime. In case of permission denial, this function will eventually trigger a runtime error. A proper usage of checkSelfPermission() is shown in Listing 2. The developer is handling the user's binary response to the permission request. If it is granted then the app can eventually access the permission and use it to fulfill its functionality.

The method *onRequestPermissionsResult()* can be overridden to handle the users response to Android's permission request. Not overriding this method is not a permission smell, but it helps from a usability point of view as an appropriate message can be shown to the user or functionality disabled if the user does not approve a permission request. Further examined use cases, relevant standards documentation and initial results are available on the project's website: *https://p-lint.github.io*.

```
void someMethod(){
  if (ActivityCompat.checkSelfPermission(this,
      Manifest.permission.READ_CALENDAR)
              != PackageManager.PERMISSION_GRANTED
                  ) {
      // Calendar permission has not been granted.
          Request Calendar Permission
      ActivityCompat.requestPermissions(this, new
          String[]{Manifest.permission.READ_CALENDAR
          },PERMISSION_READ_CALENDAR);
  }
  else {
    // Calendar permissions is already available.
        Start using the Calendar API
  }
}

void onRequestPermissionsResult(int rc, String[] p,
    int[] grantResults){
  if (grantResults.length > 0
              && grantResults[0] == PackageManager
                  .PERMISSION_DENIED) {
      // Handle the permission deny action
 }
 else{
    // Use the Calendar
 }
 }
```

Listing 2. Correct Use of CheckSelfPermission

III. Enabled Research

Although our findings are preliminary, P-Lint has already yielded some interesting results. For instance, when investigating 40 Android apps (dataset available online), we found 79 instances of the apps not utilizing the *checkSelfPermission()* function, which goes against the Android best practices guide [1] and could potentially lead to functional issues. P-Lint may be used in future research studies on large numbers of popular Android apps to answer some of the following research questions:

1) Are developers properly adhering to the defined best practices [1] for permission?
2) Are developers properly using updated functionality such as *shouldShowRequestPermissionRationale()* and what rationale are they providing? Does the rationale make sense and properly reflect the permission?
3) How frequently are developers requesting permissions, when they should have been using intents?
4) Do apps properly utilize the *checkSelfPermission()* method when permission requests are made
5) Are custom permissions being properly used?

References

[1] Permissions best practices. https://developer.android.com/training/permissions/best- practices.html.

[2] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

[3] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.