Week 2: Empirical Analysis of F-Droid

Nate Bachmeier

TIM-7140:Software Engineering

March 7, 2021

Northcentral University

# Empirical Analysis of F-Droid

F-Droid is a self-described installable catalog of FOSS (Free and Open Source) apps for the Android platform (F-Droid, 2021). There are roughly four thousand applications within their collection. Each project’s entry contains downloadable links to source repositories and official milestone releases. Numerous researchers are leveraging this information to publish papers on trends across the mobile community.

While these efforts shine light into the problem space, they are not complete. However, F-Droid represents 0.1% of the size of Google Play Store’s three million apps (Statista, 2021). Comprehensively analyzing this fraction generates enormous data volumes, requiring significant processing power, which introduces an additional sampling layer. Further, Google Play Store is the defacto solution for mainstream professional development, and excluding this population entirely creates a selection bias. Mechanisms need to exist for identifying and bridging these empirical gaps.

# Literature Review

Krutz et al. (2015) collected and analyzed metadata about 4416 versions of 1179 F-Droid projects. They used a series of static analysis tools (see Table 1) to populate an SQLite database. Academic lesson plans continue to incorporate these results, but they are not actively maintained.

Table 1: Static Analysis Tooling

|  |  |
| --- | --- |
| Tool | Description |
| Stowaway | Static analysis tool for finding under/over permissions |
| Androrisk | Androguard reverse engineering tool. Calculates risk indicators of an app |
| Sonar | Source code analyzer covering seven axes of code quality (architecture and design, comments, coding rules, potential bugs, complexity, unit tests, and duplications) |
| FindBugs | Static analysis tool for finding Java issues |
| Checkstyle | Java source analysis tool |
| PMD | Identifies maintainability risks within a codebase |
| Git | Software versioning solution with revision log |

## Identifying Research Questions

The dataset contains several data points regarding permission misuse, code quality, and the breadth of contributors. There is sufficient data to examine:

1. Does a correlation exist between app categories and permission misuse
2. Does a correlation exist between security and code quality
3. Does a correlation exist between commit counts and permission misuse

## Extending the Database

The data set follows a normalized schema to reduce the physical file size and promote consistency. This analysis adds several SQL views for ease of exploration (see Table 2).

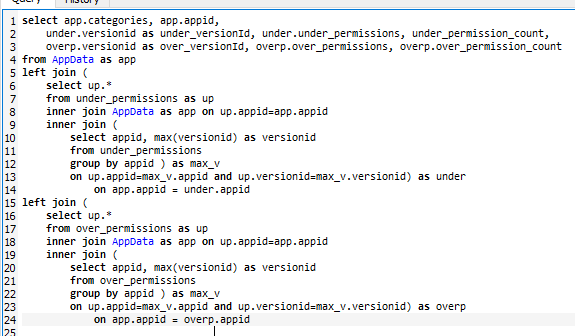
Table 2: Custom SQL Views

|  |  |  |
| --- | --- | --- |
| View Name | Description | Joins |
| over\_permissions | Unused permissions detected by Stowaway | Version, Permission, and OverPermission |
| under\_permissions | Missing permissions detected by Stowaway | Version, Permission, and UnderPermission |
| code\_risks | Combines Sonar code metrics and Androrisk score | Version, Vulnerability, and CodingStandard |
| app\_committers | App-level aggregate counts of committers | AppData and GitHistory |
| result\_set | Denormalizes per version data points into wide rows | AppData, Version,  code\_risks, app\_committers, over and under permission |

## R1: Category and Permission Misuse

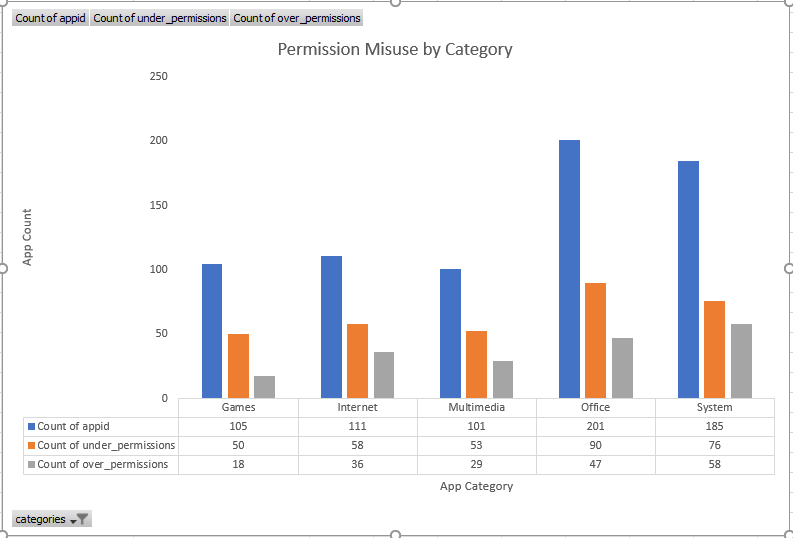
The first research question attempts to find a correlation between app categories and permission misuse. Stowaway analyzed 1621 versions of 853 sampled apps with static analysis of the Android Packages (APK) files. Since Stowaway is no longer maintained, it would be challenging to examine the remaining 63% of versions retroactively (Chromium, 2021). Instead, answers to this question only consider the most recent per-app scan result. Next, a filter retains only Games, Internet, Office, Multimedia, and System categories (835 of 853 entries). After removing erroneous and single-valued clusters, the data preparation is complete (see Figure 1).

Figure 1: Categorical Permission Misuses



Using a pivot chart suggests that most nearly between 41 to 52% of apps miss at least one required permission. Meanwhile, between 17 to 32% have too many permissions. Internet and Systems are the worst offenders due to frequently missing contacts, wake lock, and audio rights. These same categories are overprivileged with wifi, internet, and storage access. However, these are meaningless statistics because not all Android permissions are created equal. In 2017, Google restructured the permission system to consist of standard and dangerous rights. When an application is over permissive with non-dangerous rights, it does not increase the attack surface.

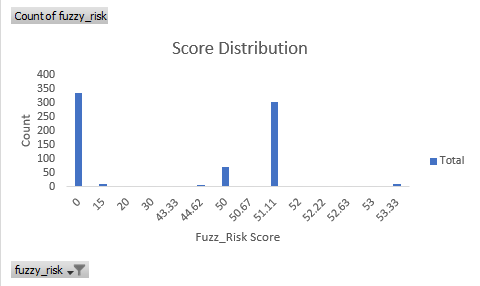
Figure 2: Permission Misuse by Category



## R2: Security and Quality Metrics

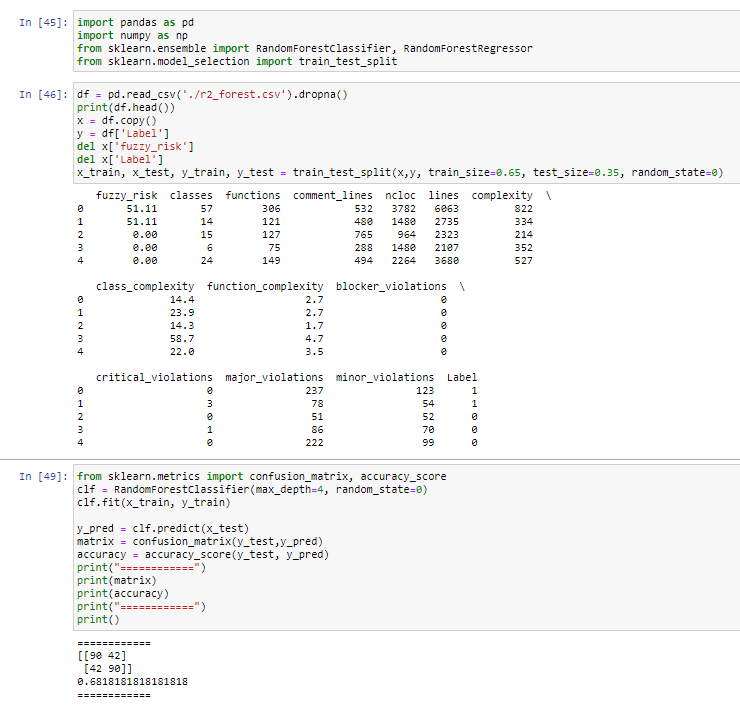
The second research question seeks a correlation between the Androrisk and Sonar results. Alenezi & Almomani (2018) address a similar problem using a random forest classifier. That work first collects twenty-one code metrics such as class complexity, duplicate blocks, and total lines of code. They combine these features with the Androrisk security score to empirically show that poorly documented complex code contains more security issues.

Figure 3: Fuzz Score Distribution



Analogously, exporting the code\_risks view into Comma Separated Value (CSV) file allows Sci-Kit Learn to process the data. This process began with removing identifier columns and scoping the dataset to twelve code metrics for 753 apps. A binary classifier is ideal for the score distribution (label = score > 30). Next, training a random forest binary classifier used 65% of the records. The trained model is 68% accurate with 90% precision and recall metrics. While there is room for improvement, these initial results suggest it is possible to predict Androrisk scores from code complexity. This characteristic infers that a correlation between the two systems must exist.

Figure 4: Training the Random Forest



## R3: Breadth of Committers and Permission Misuse