# A Thorough Security Analysis of a Third-party Android Application Marketplace

Yuchen Huang*
Case Western Reserve University
10900 Euclid Ave
Cleveland, Ohio
yxh641@case.edu

Taylor Smith†
Case Western Reserve University
10900 Euclid Ave
Cleveland, Ohio
tps45@case.edu

## ABSTRACT

The ecosystem of Android applications is thriving because there is no limitation on the source of the applications. Users could download applications from either the official app marketplace or alternative third-party marketplaces. Compared with the official marketplace, a third-party one has relatively more tolerable regulations and a simplified procedure to release applications. This convenience leads to a large group of developers and users active in the third-party marketplaces. Due to the same reasons, the security and quality of third-party marketplaces, however, is disputed. In this paper, we analyze the security features of a third-party Android application marketplace, Aptoide. We discuss the quality of third-party marketplace from two perspectives: 1, the authenticity of metadata from the marketplace. 2, the user-experience and the security of platform-occupied applications. We have three cardinal discoveries: 1, Application scores on the third-party marketplace are manually manipulated. 2, Applications from the third-party marketplace do not, as some article claimed, abuse the dangerous permissions. 3, The quality of third-party platform-occupied applications is obviously lower than the quality of applications from the official marketplace.

## Keywords

Android; App marketplace; Third-party

## 1. INTRODUCTION

The iOS and the Android are two biggest players in the family of modern smartphone operating systems. As a closed-source system powered by Apple, the iOS only supports devices from Apple's family. Applications running on iOS can be installed only from the official App marketplace. Different from iOS, the Android advocates a philosophy of liberalism. The Android is an open-source system primarily maintained by Google. Users can download Android applications from any sources, as they are not limited to the official marketplace. This characteristic leads to the emergence of a large group of active developer's communities around Android.

Users can choose any sources to download an Android application. Generally, there are three principal sources to achieve Android applications: 1, The official marketplace. 2, The manufacturer marketplace. 3, The third-party marketplace. The only official Android application marketplace is the Google Play Store which is pre-installed by the original Android systems. Also, some manufacturers choose to use their own marketplace instead of–or in addition to–Google Play. Either manufacturer marketplaces or the Google Play usually have restricted regulations and a relatively complex security verification procedure to release applications. Third-party marketplaces are, however, different from the previous two in the way that third-party marketplaces allow applications in the gray area including the jailbreak applications, illegal applications, or other regulated applications which are not allowed or are hardly allowed to be released on the official marketplace or manufacture marketplaces.

The quality of third-party marketplaces are disputed. On the one hand, the third-party marketplaces are prosperous; a large number of independent developers release applications on third-party marketplaces instead of on the official one. Users also expect to download free or exotic applications from third-party marketplaces. On the other hand, the prosperity of third-party marketplaces is somewhat relying on their flexible regulations. This flexibility, however, further benefits attackers. An attack could stealthily release malicious applications in third-party marketplaces. Moreover, the third-party marketplaces, unlike the official one, are not bound by the price of reputation. Some third-party marketplaces might manipulate data or, in addition, use fake data to make their statistics pretty.

To study the quality of third-party marketplaces, we analyze one of the largest third-party marketplaces, Aptoide, in this project. We performed a large scale of data collection. In total, we collected textual metadata of 282,905 applications from the Aptoide and 205,343 applications from the Google Play. By analyzing this data, we evaluate the quality of Aptoide from two perspectives: 1, the authenticity of metadata on the marketplace. 2, the quality of platform-occupied applications. In addition, we recruit a sentiment analysis technology to evaluate the user's reviews. By doing so, we build an estimation of the degree that users like or

*Graduate student in Computer Science department
†Bachelor student in Computer Science department

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

dislike an application.

The paper is organized as follows: In section 2, we reviewed all necessary knowledge regarding our study. We introduce our approach in the section 3. Section 4 gives our results and corresponding analysis. In section 5, we present related studies. We conclude our study in section 6.

# 2. BACKGROUND

In this section, we discuss Android system and its security features. We assume readers already have a certain level of knowledge of the Linux. We mainly concentrate on introducing the improvement and modification of Android over the Linux.

## 2.1 Android System

The kernel of Android is modified based on a long-term support Linux kernel. This allows the third-party vendors to make their own device compatible with Android by replacing the driver programs in Linux kernel. On the top of the Linux kernel, Android has a multi-layered architecture as shown in Figure 1. The Hardware Abstraction Layer (HAL) is right above the kernel layer. The HAL standardizes interfaces of the hardware capability. Thanks to the standardization, an Android application can work with different devices without any modifications. The Android Runtime (ART) is built on the HAL layer and is a critical component of the Android system, managing runtimes of all Android applications. The ART creates and allocates each application a separate environment with an independent file system. This separation protects the system and benign applications from the interferences of rouge, corrupted, or abnormal applications. Additionally, a group of native C/C++ libraries is on the same layer as the ART. These libraries provide native C/C++ interfaces for compute-intensive functions such as OpenCV, OpenGL, etc. The Java API Framework layer is above the layer of ART and C/C++ libraries. This framework provides the entire feature-set of Android OS, coming with all APIs form the building blocks of various function module such as Activity Manger and Content Provider.

## 2.2 Permissions

Android uses the permission system to manage the application's accessibility to some system resources such as camera, microphone, and SMS data. As shown in Figure 2, Android has more than a hundred permissions [4]. Application developers need to announce required permissions in a manifest file. Using restricted APIs without authorized permissions would lead the system to throw an exception. Theoretically, applications cannot override the permission system without the root priority. Some hardware vendors, however, might leave backdoors in system opening to their own applications. Such backdoor APIs cannot be found in documents but could be found by reverse-engineering the system code [2]. If backdoor APIs are utilized by an attacker, the malware could achieve the root privilege. This issue can be fixed by prohibiting unsafe APIs. In general, the permission system can effectively prevent unauthorized application from accessing restricted system resources.

## 2.3 Android Application Market

Unlike the iOS, which only allows installation from the official marketplace, the Android allows file installations. This characteristic stimulates the vitality of the third-party application marketplace. Third-party marketplaces provide a relatively relaxed environment from developers. Users may find applications that cannot be found on the official marketplace or manufacture marketplaces. Some third-party marketplaces even allow a free version of illegal copies of other paid applications. Independent developers might release applications only on the third-party marketplaces to avoid an intricate releasing procedure carried out by the official marketplace. Meanwhile, the lower bar of third-party marketplaces also benefits attackers. Repackaging attack is the most common way to lure users download malicious applications. An attacker could also use fake applications to steal users privacy.

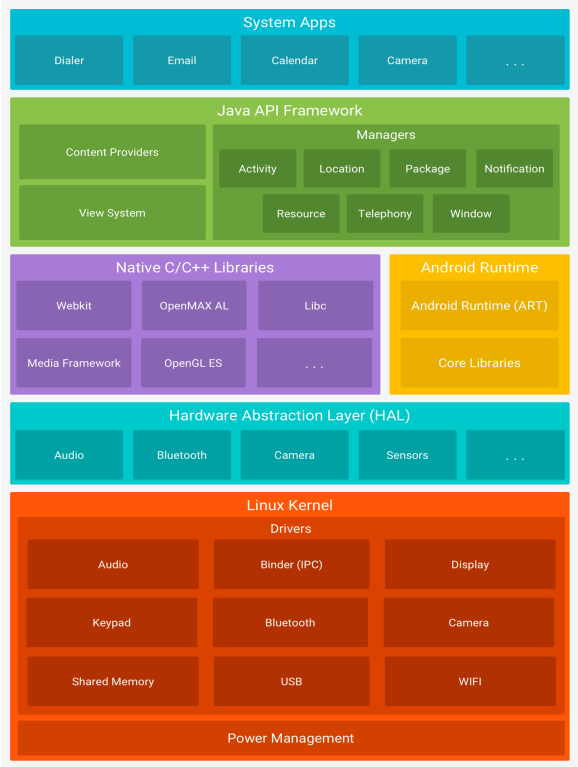


**Figure 1: The Android software stack**

# 3. APPROACH

In this section, we introduce our methodologies. Section 3.1 gives the overview of our data collection methods. In section 3.2, we discuss the way that we analyze the user reviews. Section 3.3 describes the way that we evaluate the usage of dangerous permissions.

## 3.1 Data Collection

In our project, we primarily use data from the Aptoide, but also collect data from the Google Play. The purpose of this project is to evaluate the quality of the Aptoide to reveal some facts regarding the third-party marketplace, so we sometimes make a comparison between the Aptoide and the Google Play. We first use a crawler to collect the data from the Aptoide which are ordered by the popularity. Then, we search the Google Play with applications that we met on the Aptoide. In total, we collected 282,905 records of applica-

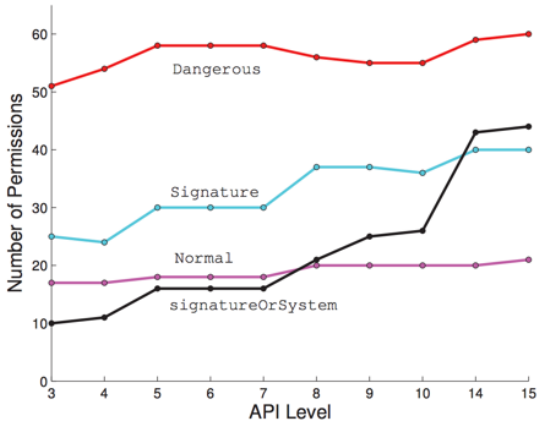


**Figure 2: Protection Levels, e.g. Normal, Dangerous, Signature, signatureOrSystem, evolving over API levels.**

tions from the Aptoide, and 205,343 records of applications from the Google Play. It is noteworthy that 77,562 applications, which occupies 27% of total applications that we met during the data collection, only can be found on the Aptoide but not on the Google Play.

### 3.1.1 Crawler of the Aptoide

The Aptoide provides both an application interface and a web interface. We design a crawler to collect data from the web interface of the Aptoide. Before massively crawling the Aptoide, we analyze the structure of the Aptoide website. We find a rank page regarding the application popularity under this URL: https://en.aptoide.com/apps/local/more?offset=0. In this URL, the parameter $offset$ can be used to control a range of ranks of applications to be presented on this page. By gradually increasing the value of the offset, we achieve a list of applications ordered by their popularity.

The next task is to extract the textual metadata from application pages. In the page of each application, we found that most of the textual metadata is directly rendered by the HTML. This is quite different from what we meet in the Google Play where most of textual metadata is not directly presented on the page, and instead are translated by a javascript after the page is loaded. Therefore, we use a list of regular expressions to extract all textual metadata from application pages of the Aptoide. As shown in Table 1, we finally collect three types of metadata, including the application metadata, the developer information, and the platform metadata.

### 3.1.2 Crawler of the Google Play

The crawling strategy for the Google Play is different from the Aptoide, because we only use the data of Google Play as a benchmark. Therefore, we were trying to find only applications that we saw in the dataset of the Aptoide from the Google, instead of iterating the rank of the Google Play. We first analyze the structure of the web interface of the Google. We find that the Google Play organizes application pages based on package names. For example, all application page's URLs are the same format: https://play.google.com/marketplace/apps/details?id=xxx, except for a different value of parameter *id*, referring to the package name of different applications. Therefore, we prepare a long list of package names of all applications that we have previously collected from the Aptoide. Then, our crawler iteratively tries each Google Play's URL which is plugging in a package name from that list.

**Table 1: Category of Textual Metadata Collected from the Aptoide**

| Application | Developer | Platform |
|---|---|---|
| App Name | Developer Name | Reviews |
| Package Name | Organization | Download Count |
| Version | Locality | App Description |
| Size | Country | Score |
| Compatibility | City | |
| Support CPU | | |
| Permission | | |
| Category | | |

## 3.2 Sentiment Analysis

User reviews reflect the real users' satisfaction toward an application. Compared with scores, user reviews are insusceptible to the forge. Inherently, a bad application is likely to be negatively reviewed, while a good application is likely to be positively reviewed. In our project, we analyze the sentiment of user reviews for following purposes: 1. We compare sentimental scores of user reviews with application scores and see whether both scores are matched. This examination could reveal a potential manipulation over application scores. 2. We use sentimental scores, in addition to application scores, to evaluate user's degree of satisfaction toward a certain group of applications. For example, we wish to know whether or not users like applications from the Google Play Store as much as applications from the Aptoide.

To analyze the sentiment of textual metadata, we use a tool which is introduced by this paper [3]. The tool tags a textual metadata with three possible labels: Positive, Neutral, and Negative, along with the label, a confidence value would be also given. In our project, we viewed the value of the confidence as a degree of the intensity of the corresponding sentiment presented by a certain textual metadata. A textual metadata contains sentimental words, such as love, like, dislike, etc., result in a high value of confidence.

An application usually contains many reviews. Each review consists of two parts: a title and a textual content body. To generally evaluate the overall sentiment score of a given application, we design an algorithm to cumulatively calculate all reviews of this application. We define the sentimental score of a textual metadata as a factor multiplied by the confidence value given by the tool previously mentioned. The factor is decided by the label generated by that tool as well as whether the factor is equal to -1 if the label is the Negative, equal to 1 if the label is the Positive, and equal to 0 if the label is the Neutral. For each review, the sentiment score of the title part has a weight of 0.6 and the sentiment score of the content part has a weight of 0.4, so the score of a review satisfies the equation 1:

$$score_{(review)} = 0.6 * score_{(title)} + 0.4 * score_{(content)} \quad (1)$$

We assign 0.6 to the weight of titles because we observe that

users usually use concise and direct words on the review title to summarize their reasons. A review can be viewed as positive if its $score_{(review)}$ is greater than zero. In the same manner, a negative review is defined if its $score_{(review)}$ is less than zero. Eventually, we claim an application is positively reviewed if the number of positive reviews is greater than the number of negative reviews, and vice-versa.

### 3.3 Dangerous Permission Evaluation

As we mentioned before, the usage of permissions somewhat reflects the risk of an application. Using too many dangerous permissions will not only raise the suspicion but also vitiate the opportunity of an application to pass the security check. In our project, we wish to evaluate the usage of dangerous permissions. We extract a list of sensitive permissions from the paper of AutoCog including:

1. WRITE_EXTERNAL_STORAGE,
2. ACCESS_FINE_LOCATION,
3. ACCESS_COARSE_LOCATION,
4. RECEIVE_BOOT_COMPLETED,
5. GET_ACCOUNTS,
6. CAMERA,
7. WRITE_SETTINGS,
8. READ_CALENDAR,
9. RECORD_AUDIO,
10. READ_CONTACTS,
11. WRITE_CONTACTS.

Given an application, we match its permissions with this list, and see how many sensitive permissions are used. However, using the sensitive permissions does not always mean that the user is at risk, but also could indicate the complexity of an application. An application with a comprehensive function set might result in this application using various types of permissions.

## 4. RESULT AND ANALYSIS

In this section, we systematically introduce our statistical result, and also, we give our analysis and hypothesis on these data.

### 4.1 Application Scores

The score of applications is critical information to tell the quality and the overall satisfaction. In addition, the score is usually viewed as a key feature, for a marketplace, to recommend applications to users. Using fake scores might result in users downloading low-quality applications, or even, malicious applications. Therefore, the authenticity of the score is a decisive feature to evaluate the security of application marketplace.

Both the Aptoide and the Google Play adopt a scoring system of a scale of five stars where the best is five stars and the worst is one star. We project the scale of five stars to a scale of 100 points, and respectively calculate the average number of points that applications from each marketplace achieve. As shown in Figure 3, the average scores of applications from both marketplaces are around 81 points which are equal to four stars in the scale of five stars. There is no big difference between the average scores of the Google Play and the Aptoide.

Then, we generate the distribution of a number of applications to the scores. As Figure 4 shows, the X-axis refers to scores, and the Y-axis refers to the percentage of applications falling in a certain score. The green curve represents

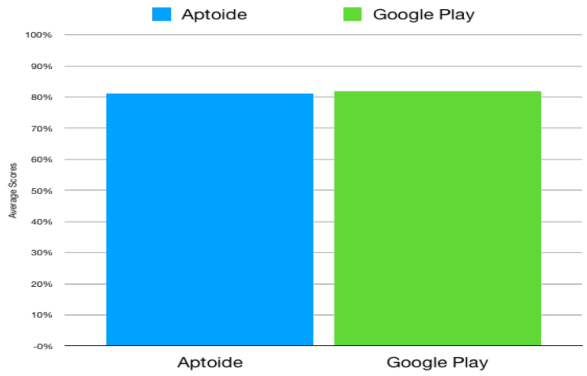


**Figure 3: Average Application Scores**

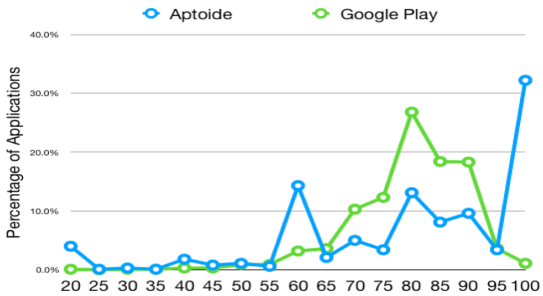


**Figure 4: Application Distribution over Scores**

the distribution of the Google Play, and the blue curve represents the distribution of the Aptoide. Here, we observed an unusual distribution from the Aptoide dataset. As you may find, both curves have a normal distribution around a median value of 80. However, the curve of Aptoide has two abnormal peaks at 60 and 100 corresponding to the 3 stars and 5 stars in the scale of five stars.

It is a strong signal that the scores of Aptoide suffer from manipulation. This manipulation could either be made by the Aptoide with the purpose to make the statistics look pretty, or by the developers to raise the ranks of their applications. In either case, the Aptoide should take responsibilty for the failure of providing authentic scores to users.

### 4.2 Application Reviews

The application review is another key piece of information to tell users the advantages and the disadvantages of an application. Compared with the score, the review is insusceptible to the forge. We analyze user reviews for two reasons: First, we can match the statistics of user reviews with the statistics of scores to reveal the potential manipulation of the scores. Second, we can compare different sets of applications to reveal a potential difference in the quality of different sets of applications. Here, we group applications into two sets: one set contains applications on both the Google Play and the Aptoide, and another set contains applications only on the Aptoide.

We use the previously introduced sentiment analysis method to process user reviews. For each application, we can finally achieve a label telling that the application is positively, neutrally, or negatively reviewed. As shown in the Figure 5, 26% of applications only on the Aptoide are positively reviewed,

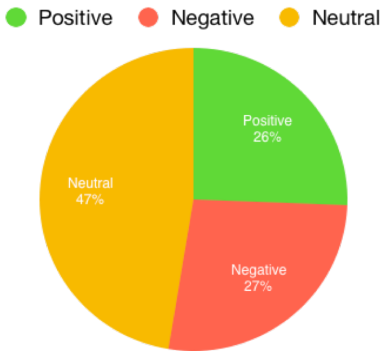


**Figure 5: The Satisfaction over the Applications only on the Aptoide**

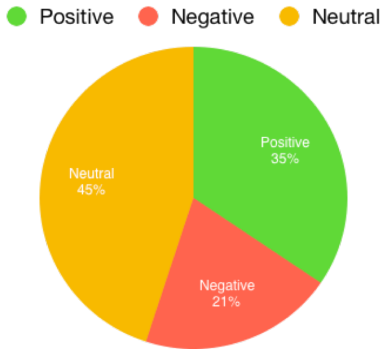


**Figure 6: The Satisfaction over the Applications on both marketplaces**

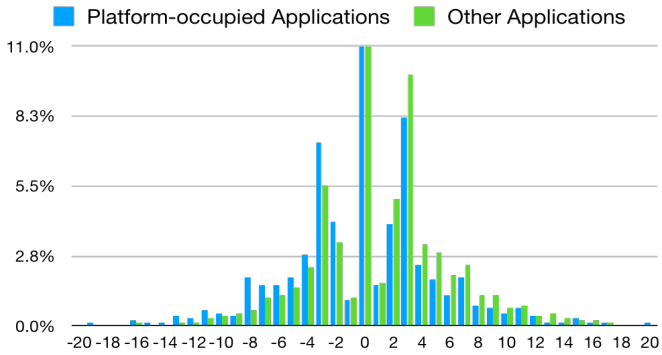


**Figure 7: The Satisfaction Distribution**

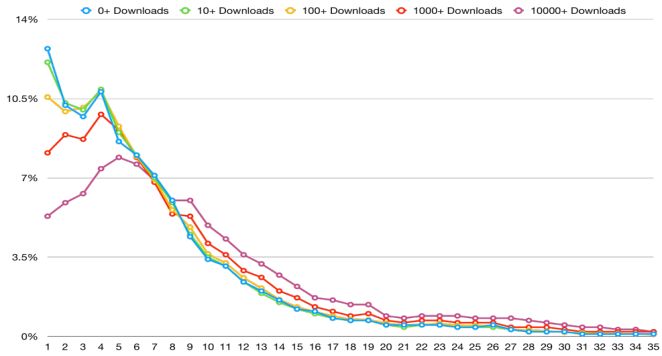


**Figure 8: The Distribution of Permission Usage of Applications on both marketplaces**

while 27% of applications are negatively reviewed. As shown in the Figure 6, 35% of applications on both marketplaces are positively reviewed, and only 21% of applications are negatively reviewed. This result shows a preliminary result that the overall quality of applications on both marketplaces is obviously better than those applications only on the Aptoide.

Using a single label to summarize users' satisfaction may lose some important information. For example, some applications may be worse or better than others, but all these differences cannot be distinguished by using only labels. To solve this issue, we also project the degree of satisfaction of applications into a range of degree from -20 to 20 in which the most unsatisfied degree is the -20 and the most satisfied degree is the 20. By doing so, we generate a distribution of applications to the degree of satisfaction. As shown in Figure 7, the X-axis represents the degree of satisfaction, and Y-axis represents the percentage of applications with a certain degree of satisfaction. The reason for the peak at 0 is that there are many non-English reviews, and for these reviews, the sentiment analysis tool will always give a neutral label. We observe two normal distribution patterns: one distributes around the -3 and another distributes around 3. These normal distributions show that most of application reviews are unambiguous. Users comment on an application for a certain reason, either they like the application or dislike the application. In contrast to the score distribution, there is no weird peak among the statistical data of user reviews. Therefore, we have a further confirmation that the scores of the Aptoide are manipulated.

Furthermore, we notice that the satisfaction of the applications only on the Aptoide is clearly worse than the satisfaction of applications on both marketplaces. This shows that the overall quality of those third-party-only applications is not as good as applications on both the official and third-party marketplaces.

## 4.3 Permissions

As we mentioned in the previous section, the usage of permissions would reveal some information regarding the complexity and risk of an application. In our project, we analyze the difference of the permissions usage between the set of applications on both marketplaces and the set of applications only on the Aptoide. In addition, we also analyze the effect of the popularity of applications on the permission usage. Figure 8 shows the distribution of permission usage of applications on both marketplaces, and the Figure 9 shows the distribution of permission usage of applications only on the Aptoide.

From both figures, we see that most of the applications use less than 20 permissions, but we also observe few applications use more than 100 permissions including some manufacturers' permissions. We observe some interesting facts by comparing two figures:

1. *Figure 8 has relatively more flat curves when the download count is greater than 10000.* One explanation is that the applications only on the Aptoide, even among those famous ones, are likely to be applications of medium or small size, which have relatively simple functions and require fewer

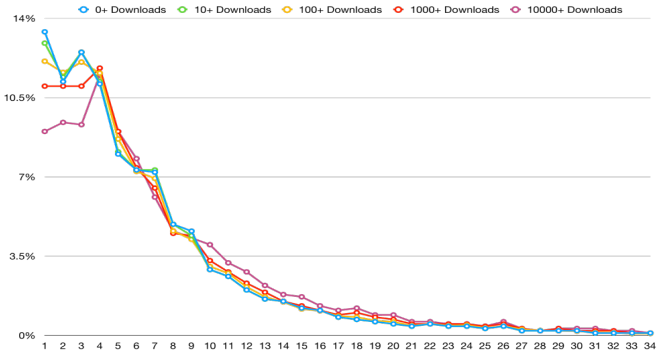


**Figure 9: The Distribution of Permission Usage of Applications only on the Aptoide**

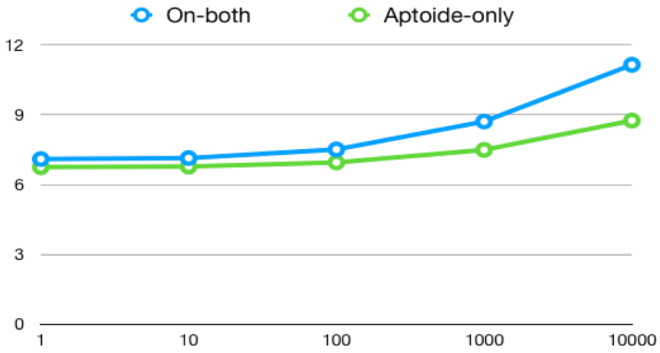


**Figure 10: The Distribution of Permission Usage of Applications on both marketplaces**

permissions.

2. *Figure 8 has sub-peak at 4, while Figure 9 has sub-peak at 3.* Curves from both figures are going down, up, and then down again with the number of permissions increasing. We observe that the sub-peak of the set of applications on both marketplaces is located at 4, while the corresponding peak of the set of applications only on the Aptoide is at 3. We do not have a convincing explanation for this phenomenon. However, one hypothesis is that the applications on both marketplaces are more likely to use a certain set of permissions, which prefer results with different permission usages.

To independently analyze the effect of the popularity of an application on the permission usage, we generate the Figure 10 where the X-axis represents the number of downloads, and the Y-axis represents the average number of permission usages. In this figure, we see that the increase in the number of permissions correlates with the increase in the number of downloads. This is because the popular applications are likely to be more complex than other applications, so these popular applications use more permissions than average. Also, we observe that the curve of the set of applications on both marketplaces is increasing faster than the curve of the set of the application only on the Aptoide. This phenomenon reinforces our statement that the applications only on the Aptoide are likely to be medium or small size applications.

Using the number of permissions to evaluate the risk of applications is somewhat misleading because a benign application might use many harmless permissions but a malicious application only a few dangerous permissions. In our project, we also analyze the dangerous permission usage. As shown in Figure 11, we calculate the average number of dangerous permissions. The average numbers of used dangerous permissions in both sets of applications are around 2.4. Therefore, there is no strong signal showing that the applications only on the third-party marketplace abuse dangerous permissions.

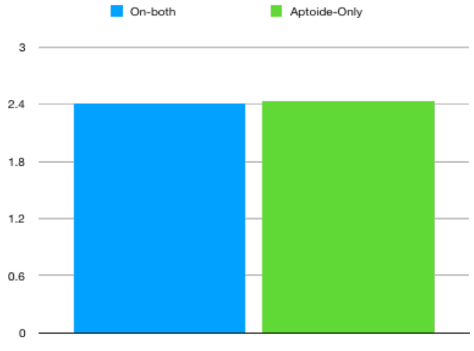


**Figure 11: Average Number of Dangerous Permissions**

## 5. RELATED WORK

Zhou et al., performed a systematic study on six popular Android-based third-party markets to detect repackaging applications [6]. To find the similarity between repackaging application and original application, they recruit the DroidMOSS that is working based on the fuzzy hashing technique. They finally detect 13% of applications in this markets are repacking applications.

Yang et al., introduces an approach to against applications money-stealing applications on the alternative application markets[5]. Their approach relies on detecting two sensitive behaviors: the hardcoded exfiltration and the notification suppression.

Guzman et al., applied natural language processing technique to filter, aggregate, and analyze user reviews, and then use sentiment analysis and topic modeling technique to generate a meaningful high-level features [1].

## 6. CONCLUSION

In our project, we perform a thorough analysis of the third-party application marketplace, Aptoide. We show that the scores of the Aptoide are manipulated. We also discuss the quality of the set of applications only on the Aptoide. Our results clearly show that this set of applications has a relatively lower quality than other applications. Additionally, we disprove the rumor that applications only in the third-party marketplaces abuse dangerous permissions.

In conclusion, the authenticity of third-party application marketplaces are suspicious, and the overall application quality only on third-party marketplaces are obviously lower than the quality of applications on the official marketplace. Although third-party marketplaces provide a flexible environment for developers, these marketplaces still need to maintain a good self-discipline and improve techniques to avoid score manipulations.

# 7. REFERENCES


[1] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 153–162. IEEE, 2014.

[2] A. Moulu. Android oemâĂŹs applications (in) security and backdoors without permission.

[3] V. Narayanan, I. Arora, and A. Bhatia. Fast and accurate sentiment classification using an enhanced naive bayes model. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 194–201. Springer, 2013.

[4] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 31–40. ACM, 2012.

[5] C. Yang, V. Yegneswaran, P. Porras, and G. Gu. Detecting money-stealing apps in alternative android markets. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 1034–1036. ACM, 2012.

[6] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326. ACM, 2012.