Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems | School of Computing and Information Systems

10-2016

Dissecting developer policy violating apps: Characterization and detection

Su Mon KYWE
Singapore Management University, monkywe.su.2011@phdis.smu.edu.sg

Yingjiu LI
Singapore Management University, yjli@smu.edu.sg

Jason HONG
Carnegie Mellon University

Yao CHENG
Singapore Management University, ycheng@smu.edu.sg

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the Information Security Commons

Citation

KYWE, Su Mon; Yingjiu LI; HONG, Jason; and CHENG, Yao. Dissecting developer policy violating apps: Characterization and detection. (2016). *MALWARE 2016: Proceedings of the 11th International Conference on Malicious and Unwanted Software: Fajardo, Puerto Rico, October 18-21*. 10-19.
Available at: https://ink.library.smu.edu.sg/sis_research/3381

This Conference Proceeding Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Dissecting Developer Policy Violating Apps: Characterization and Detection

Su Mon Kywe[1], Yingjiu Li[1], Jason Hong[2], Cheng Yao[1]
{*monkywe.su.2011, yjli, ycheng*}*@smu.edu.sg*[1]
School of Information Systems, Singapore Management University[1]
{*jasonh*}*@cs.cmu.edu*[2]
School of Computer Science, Carnegie Mellon University[2]

Abstract

To ensure quality and trustworthiness of mobile apps, Google Play store imposes various developer policies. Once an app is reported for exhibiting policy-violating behaviors, it is removed from the store to protect users. Currently, Google Play store relies on mobile users' feedbacks to identify policy violations. Our paper takes the first step towards understanding these policy-violating apps. First, we crawl 302 Android apps, which are reported in the Reddit forum by mobile users for policy violations and are later removed from the Google Play store. Second, we perform empirical analysis, which reveals that many violating behaviors have not been studied well by industry or research communities. We discover that 53% of the reported apps are either copying popular apps or violating copy-rights or trademarks of brands. Moreover, 49% of reported apps are violating ads policies by sending push notifications, adding homescreen icon and changing browser settings. Only 8% show malware-like behaviors, such as downloading malicious files to users' mobile phones. Based on our empirical analysis results, we extract 175 features for differentiating bad apps from benign apps. Our features cover use of brand names and other keywords, third-party libraries, network activities, meta data, permissions, and suspicious API calls originated from third-party libraries. We then apply 10 machine learning classifiers on the extracted features to detect reported bad apps. Our experiment result shows that the best algorithm can detect them with 86.80% true positive rate and 13.6% false positive rate. On the other hand, the same samples of policy violating apps are detected by VirusTotal with true positive rate of 55.63% and false positive rate of 17.48%.

1 Introduction

Google Play store is infected with various types of bad apps, including malware apps, privacy-violating apps, and repackaged apps. Google has been trying to take down these apps everyday with the help of anti-malware technologies, such as Google Bouncer, and inputs from researchers and users. Previously, researchers have tried to understand malware apps' behaviors by painstakingly collecting real-life malware app samples [34]. They have also proposed various prevention or detection techniques for malware apps [11] [1] [21], privacy-preserving apps [28] [35] [27] [26] [20] [24], and repackaged apps [29] [33] [23] [5] [9] [30]. However, many apps behave in a manner that is undesirable, and yet less serious than these apps. For instance, some apps redirect users to share about the apps on Facebook page. Some are spams. Some apps simply lack proper functionalists and qualities. These apps violate Google Play developer policies but little has been discovered about the overall picture of these apps. Therefore, we attempt to answer the following questions in our paper: What categories of bad apps are the most common ones on Google Play store? What characteristics and behaviors make them reported by users and removed from the market? Can the existing anti-virus solutions be used in detecting them?

Our paper makes the following contributions: (1) We collect a set of real-life apps that are reported by users, and later taken down by Google Play store. The lack of sample set has been deterring researchers from creating solutions and evaluating their effectiveness. Our collection of real-life bad apps can serve as a baseline for various future analysis and research (2) We perform extensive empirical analysis on the app samples we collected. Comprehending and characterizing these apps is the first step towards designing defense mechanisms against them. Our empirical analysis provides answers to various questions left unanswered by previous research (3) We use machine learning approach to detect the collected app samples. Although machine learning algorithms have been commonly applied for malware apps detection, different feature sets are required for detecting our app samples due to their unique characteristics and behaviors. In our paper, we also figure out whether the existing anti-virus solutions can effectively detect these

policy-violating apps.

First, we build an automated crawler, which collects real-life apps that violate Google Play developer policies. Google enforces these policies to maintain quality and health of mobile ecosystem as well as to provide great experience for mobile users. Our crawler crawls the posts from Reddit forum under Bad app category every 5 minutes. It obtains the links of reported apps from the posts and immediately downloads them from the Google Play store. After 3 months of automated crawling, our crawler follows the same links again and checks if the bad apps are indeed removed from the official Google Play store. In this way, we are able to collect 302 bad apps that are reported and removed from the Google Play store.

Second, we perform extensive empirical analysis on the app samples we collected. Out of 302 crawled apps, we discover that 161 apps violate intellectual property rights of other apps or brands. The most copied apps include Bejeweled Blitz, Candy Crush, Mine Craft, Angry Bird Rio, Flappy Bird, Hay Day, Fruit Ninja, Subway Surf, Construction City, Sonic Dash, Gangster Vegas, and Grand Theft Auto. The most copied trademarks or brands include Pikachu, Adobe, Pou, Mario, Disney, Mickey, Minion, Counter Strike, and Despicable Me. The violation normally takes place in apps' titles, descriptions, icons or screen-shots. We discover that 79 of them use similar titles, 76 use screen-shots that are different from in-app screen-shots, and 73 use mis-leading descriptions that are different from apps' actual functions. Many reported bad apps use misleading keywords, such as 2, II, Demo, Free, Pro, 3D, and HD, claiming that they are an enhanced version of original apps. Moreover, we discover 67 apps that claim to contain certain functions but actually do nothing. They include fingerprint scanner apps, flash light apps, font apps, wallpaper apps, bluetooth apps, volume boosters, wifi boosters, and mp3 downloaders.

Among the crawled apps, 147 apps violate ad policies. We discover 70 apps with ads that simulate the user interfaces of the apps, 54 apps that modify browser settings or add homescreen shortcuts on the users' device as a service to third parties or for advertising purpose, 34 apps that show ads outside the applications, 17 apps from which users cannot dismiss their ads without penalties or inadvertent click-throughs, and 10 apps that display ads through system level notifications. Our analysis shows that the most common violating ad libraries include startapp, inmobi, umeng, ironsource and actionbarsherlock. Moreover, we discover that 30 apps redirect users to install other apps from Google Play store or third-party markets, 16 apps violate Youtube policies by downloading videos from Youtube, and 9 apps download unwanted mp3 files. We also discover 2 apps which are automatically created by wizard services, and another 2 apps which offer incentives to rate the apps.

Third, we apply machine learning algorithms for detecting such misbehaving apps. To test the effectiveness of our detection, we apply it on 302 real-life policy-violating apps that we collected and 326 benign apps from Google Play store. We extract 175 features from the apps' use of brand names and keywords, third-party libraries, network activities, meta data, permissions, and misbehaving API calls that are originated from third-party libraries. We input the extracted features into 10 machine learning classifiers for differentiating policy-violating apps from benign apps. Three-fold cross validation is performed, where two-third of our data set is used for training and one-third of our data set is used for testing. The experimental results show that our detection algorithm can effectively detect bad apps with 86.80% true positive rate and 13.6% false positive rate. We also test our samples with VirusTotal [25], which scans the submitted apps with existing 57 anti-virus solutions. We assume that VirusTotal can detect a policy-violating app as long as one of its anti-virus software reports it as bad. In this way, we discover that the true positive rate of VirusTotal is 55.63% and its false positive rate is 17.48%. In terms of individual performance, the best anti-virus solution of VirusTotal can detect the policy-violating apps with the true positive rate of 36% only. Our research shows that despite the efforts of industry and research communities in app market regulation, the problem of policy-violating apps is still prevalent and requires attention from both industry and research communities.

The rest of the paper is organized as follows. Section 2 clarifies our data collection process. Section 3 provides our empirical analysis and findings. Section 4 describes the details of our detection mechanism, and Section 5 discusses the experiment results. Section 6 summarizes the related work and Section 7 concludes the paper.

2 Data Collection

In this section, we describe how we collect policy-violating app samples. These samples are very useful for analyzing the apps' behaviors and consequently for designing defense mechanisms and evaluating them. However, obtaining a set of policy-violating apps is not trivial. There are two challenges to it. The first challenge is noticing what apps are violating Google play policies, because users' reports to Google Play store are not available to the public. To solve this, we seek to a public forum, Reddit, for such app reports by users. The second challenge is that there is a small time frame to crawl or download policy-violating apps from Google Play, once the report has been made. To solve this, we develop an automated crawler for downloading bad apps and creating a database of bad apps. We plan to make the dataset available to the public, after this paper is published.

To find users' reports, we first crawl posts from Reddit forum under the `https://www.reddit.com/r/BadApps` URL. This URL is a subReddit (i.e. subforum) used to report inappropriate apps to Google. The description of the subReddit is as follows. "A subreddit to discuss and coordinate reporting bad apps to the Google Play Store. Note: A bad app refers to apps that are fake, pretending to be from different developer, harmful, are there just to serve annoying ads to you, or steal your info. An app that is just poorly made should not be posted here, they are fine." Reddit provides Application Programming Interfaces (APIs) for various functions, including messaging, editing posts and reading posts. The returned objects are in JavaScript Object Notation (JSON) format. We develop a Python crawler, which checks the BadApp subReddit forum every 5 minutes for latest users' posts and comments. Once we find a new link of reported app, we crawl the apps' metadata, as well as Android Application Package (APK) files from the Google Play store. The crawling continues for over 3 month period. After that, we check the same links of the apps again, and see if they are indeed removed from the Google Play store. In this way, we crawl about 302 policy-violating apps.

The second set of our data is benign apps from Google Play store. We randomly download 326 benign apps, which exist for at least 3 months from the Google Play store with an unofficial Python API.

3 Empirical Analysis

In this section, we perform empirical analysis on our bad app samples and explain our findings on their characteristics and behaviors. To understand policy violation behaviors, we first put the Google Play's developer policies into four main categories: (1) intellectual property and deception, (2) monetization and ads, (3) spam, store listing and promotion, and (4) security and privacy. Although restricted content, such as sexually explicit content and violence, are parts of Google Play policies, we do not find any apps violating these policies. Table 1 shows the results of our empirical analysis. We discover that more than half of bad apps are violating copy-rights or trademarks and about half of bad apps are violating ad policies. Many of them also show several other bad behaviors, such as spamming, violating Youtube policies and downloading external files. The most common types of policy violations are (i) apps that use similar title to branded apps, (ii) apps that use photos or screenshots from other brands, (iii) apps that provide misleading description, (iv) apps that have little or no functions, and (v) apps that hold ads simulating the user interfaces.

3.0.1 Intellectual Property Right Violations

The first five categories of policy-violating behaviors in Table 1 are related to copy-right or trademark violations. Copying may occurs in five places of bad apps: icons, titles, screenshots, descriptions or functions. An interesting finding is that more copying apps make their icon, title and descriptions similar to those of original apps, instead of copying the exact features. Further analysis shows that reported apps tend to use the keywords, such as 2, II, Demo, Free, Pro, 3D, and HD, in their titles claiming that they are an enhanced version of original apps. Moreover, we discover that more bad apps make their icons from the screenshots included in the metadata of original apps or in-app real screenshots of original apps. However, many of them use the same screenshots that as the original apps. These findings have several implications for detecting apps that violate intellectual property rights. For instance, similarity scores between icons of copying apps, and screenshots of copied apps should play an important role in detecting such apps. We also discover that many of them uses screen-shots that are different from in-app screen-shots and mis-leading descriptions that are different from apps' actual functions. This suggests that if we can find discrepancies in these bad apps, we will be able to detect them well.

We discover that there exist not only apps which copy other original apps but also apps which violate the copyrights or trademarks of other brand or websites. For instance, some apps include intellectual properties of Pokemon or Play Station although these original companies do not have any related apps in the Google Play store. We further analyze apps and brands that these bad apps normally copy. They include Bejeweled Blitz, Candy Crush, Mine Craft, Angry Bird Rio, Flappy Bird, Hay Day, Fruit Ninja, Subway Surf, Construction City, Sonic Dash, Gangster Vegas, and Grand Theft Auto. The brands or trademarks that are violated include Pikachu, Adobe, Pou, Mario, Disney, Mickey, Minion, Counter Strike, and Despicable Me.

Interestingly, we only find 11 repackaged apps whose functions are exactly the same or similar to original apps. This shows that although bad apps are one of the main distribution channels of malware, they are only a small portion of entire policy-violating apps. Apps that claim to contain some functions but actually doing nothing include fingerprint scanner, flash light apps, font apps, wallpaper apps, bluetooth apps, volume boosters, wifi boosters, MP3 downloaders, and other music downloaders. The primary functions of some bad apps are to redirect users to other websites. We also find 22 apps which use irrelevant and excessive keywords in app descriptions. For example, some font apps use "Samsung Galaxy" keyword extensively in their descriptions to direct mobile users to their apps during the search. Such apps mostly reduce the quality of the app market.

Category	Policy Violating Behaviors	No of Apps
Icon (Copy-right)	Same as the icon of original app	4
	Similar to the icon of original app	15
	Violates copyright or trademark of original app (e.g. copying title, screenshots, in-app real screenshots of original app)	52
	Violates copyright or trademark of other brands or websites	32
Title (Copy-right)	Same as the original app	3
	Similar to the original app	79
	Violates copyright or trademark of other brands or websites	35
Screenshot (Copy-right)	Same as the original app	35
	Similar to the original app	4
	Violates copyright or trademark of original app (e.g. copying title, in-app real screenshots)	39
	Violates copyright or trademark of other brands or websites	38
	Different from the real in-app screen	76
Description (Copy-right)	Same as the original app	5
	Similar to the original app	36
	Violates copyright or trademark of original app (e.g. including other apps brand names)	41
	Violates copyright or trademark of other brands or websites	38
	Different from actual function (Misleading descriptions)	73
	Irrelevant and excessive keywords in apps descriptions	22
Function (Copy-right)	Very little or no function (e.g. blank page or a video keeps playing)	67
	Same as the original app	6
	Similar to the original app	5
	Violates copyright or trademark of original app (e.g. including other apps resources)	7
	Violates copyright or trademark of other brands or websites	19
	The apps primary function is to reproduce or frame someone elses website (i.e. webview of anther website)	19
Ads	Ads simulate or impersonate the user interface of any app	70
	Ads are displayed outside the app (Ads simulate or impersonate UI notification and warning elements of the operation system)	34
	Displays advertisements through system level notifications (Push notifications)	10
	Users cannot dismiss the ads without penalty or inadvertent click-through (e.g. exit ads)	17
	Modifies or adds browser settings or bookmarks, adds homescreen shortcuts, or icons on the users device as a service to third parties or for advertising purpose	54
Spams	Created by an automated tool or wizard service and submitted to Google Play by the operator of that service on behalf of other persons	2
	Offer incentives for rating	2
Others	Violates Youtube policies	16
	Automatically redirects users to install other apps (including other third-party market apps) from Google Play	30
	Includes buttons in apps to download other apps from Google Play	49
	Forces users to download files or install apps from sources outside of Google Play	25
	Downloads unwanted mp3 files	9

Table 1: Empirical Analysis on Policy Violations

3.0.2 Ad Policy Violations

The past studies on ad libraries have been focusing on privacy issues, such as phone ID and location collection by ad libraries. However, our empirical analysis results show a different set of policy-violating ad libraries. Our data set includes 70 apps with ads which simulate or impersonate user interfaces of the apps. We also discover 54 ad-policy violating apps that modify browser bookmarks or add homescreen shortcuts on users' mobile phones. Moreover, we find 34 apps, which display ads outside the apps using warning elements of Android. We also discover 17 apps where users cannot dismiss ads without penalty or inadvertent clickthrough and 10 apps where ads are sent via system notifications.

The most common policy-violating ad libraries are Start App, Inmobi, Umeng, and IronSource. Start App library shows interstitial ads, splash ads, exit ads, native ads and reward ads. Inmobi ad library includes interstitial ads and native ads. Umeng ad library sends ads via system notification bar, downloads and requests installation of new applications, and send information to a remote location about currently running applications, installed applications, and device information such as International Mobile Station Equipment Identity (IMEI), kernel version, phone manufacturer, phone model details, location (such as GPS coordinates, cell tower location), and network operator information. Ironsource library displays native ads and video ads.

In addition to ad libraries, we discover other common types of libraries among bad apps, including (i) `com.unips`, which provide live wallpaper adware, (ii) `com.monotype`, which claims to provide free font, (iii) `com.rahul`, which provides Youtube downloader, (iv) `com.unity3d` and `org.andengine`, which are game developing libraries, and (v) `io.card` credit card scanning library under the URL https://github.com/card-io/card.io-Android-SDK.

3.0.3 Spams

Some apps violate policies by applying automated app creation tools, such as `https://www.appsgeyser.com/`, or offering incentives for rating the apps. We find only 2 apps in each category. Despite this small number, we notice during crawling that some Reddit-forum users report developers who are spamming the market by creating hundreds of similar apps. Studying these spammers requires different kind of analysis on developer accounts in addition to apps themselves. Thus, we leave them as future work.

3.0.4 Others

We discover other apps that misbehaves, but Google's developer policies do not cover. They include apps that violate Youtube policies, redirect users to other apps, and force users to download unwanted files. We discover that 25 apps force users to download `mobogenie.apk` file, which enables control from a remote computer. Mobogenie may also be used to download apps, images, videos or music to mobile phones, manage SD cards, create backups on computer, and edit phone contacts. We find 30 apps, which redirect users to `spearmintbrowser.com`, which claims to provide AdBlock and build-in Flash support features. We also discover 10 apps, which download APK file named as flash player, 80 bad apps, which redirect users to download other apps from Google play and other third-party markets, and 12 apps, which connect to and grab data from Youtube. Many apps also access mobile users' accounts by obtaining authentication tokens: 118 apps access to Google account, 8 to PayPal account and 24 to Twitter account. Moreover, we find out that 29 bad apps attempt to share posts on users' Facebook account using URLs, such as https://m.facebook.com/dialog/feed?app$_id = 0link = 1picture = 2name = 3description = 4redirect_uri = 5$.

4 Detection

Our detection includes two steps: (1) extracting typical features from both bad apps and benign apps and (2) applying selected machine learning algorithms to detect bad apps. The detection can be performed by either security researchers or Google Play Store managers for vetting submitted apps before they are officially released.

4.1 Feature Extraction

We extract six groups of features from mobile apps, including the use of brand names, third-party libraries, network activities, meta data, permissions, and API calls. The features can be grouped into two. The first category includes features derived from our empirical findings, such as popular brands or app names, network activities and third-party libraries. The second category is based on behavior-based features, such as permission and API-based features. Our feature extraction is implemented in Python, and detection algorithms are run in Java. In particular, Androguard library [8] is used to reverse engineer the app codes, and extract the information about third-party libraries, network activities, permissions and API calls. To obtain third-party libraries, we first use the `dx.get_tainted_packages().get_packages()` method of Androguard to obtain package lists from apps and then, we exclude the package names of

Classes	Methods
android.app.NotificationManager	notify()
android.app.AlertDialog.Builder	show()
android.widget.Toast	show()
android.provider.Browser	saveBookmark()
android.provider.Browser	sendString()
android.content.Context	startActivity()
android.net.Uri	parse()
java.lang.ClassLoader	loadClass()
java.lang.Class	forName()
java.lang.Class	getDeclaredMethod()
java.lang.Class	getMethod()
java.lang.reflect.Method	invoke()
dalvik.system.PathClassLoader	init()
dalvik.system.DexClassLoader	init()
dalvik.system.DexFile	loadDex()

Table 2: APIs Used as Features in our Detection

the apps. To analyze the network activities, we first obtain the String values of APK files via the `d.get_strings()` method, and then, filter out the values starting with "`http://`" or "`https://`". For searching the API calls from third-party libraries, we apply `dx.tainted_packages.search_methods()` method, and determine whether they are originated from app source codes or third-party libraries' source codes.

4.1.1 Use of Brand Names and Other Keywords

We blacklist a list of popular brands and app names that we have obtained from our empirical analysis. Each brand name represents a feature for our detection. We also include other keywords, such as 2, II, Demo, Free, Pro, 3D, and HD as features. Overall, we use the following 56 words as features in our detection: 'flash', 'light', 'bejeweled', 'blitz', 'wAsk', 'racing', 'live', 'wallpaper', 'construction', 'city', 'studio', 'candy', 'bluetooth', 'free', 'game', 'sniper', 'crime', 'craft', 'mine', 'pikachu', 'font', 'farm', 'app', 'video', 'download', 'tube', 'pou', 'gangster', 'bird', 'flappy', 'subway', 'surf', 'dash', 'grand', 'theft', 'sonic', 'rio', 'ninja', 'demo', 'pro', '3D', '2', 'II', 'hay', 'day', 'flv', 'adobe', 'install', 'despicable', 'font', 'galaxy', 'monotype', 'volume', 'boost', 'mp3', and 'music'.

4.1.2 Third-Party Libraries

We blacklist the following 16 third-party ad libraries that are shown to include aggressive ad behaviors: 'startapp', 'inmobi', 'umeng', 'ironsource', 'actionbarsherlock', 'millennialmedia', 'adsdk', 'revmob', 'chartboost', 'fmod', 'furry', 'mobclix', 'appflood', 'tapjoy', 'jirbo', and 'squareup'. The presence of each library is regarded as one feature in our detection.

4.1.3 Network Activities

We maintain the following 20 blacklisted servers, and determine whether apps connect to them in their APK files: 'admob', 'gstatic', 'startappexchange', 'ad-market', 'searchresults', 'inmobi', 'umeng', 'googleapis', 'akamaihd', 'appsdt', 'spearmint-browser', 'mobilecore', 'avazutracking', 'cloudfront', 'youtube', 'rightyoo', 'iron', 'scmpacdn', 'airpush', and 'ytimg'.

4.1.4 Meta Data

From the metadata of an app, we extract the number of downloads, the app's APK file size, the number of ratings, the average star rating, the number of users rating one star, the number of users rating two star, the number of users rating three star, the number of users rating four star and the number of users rating five star. Thus, the meta data contribute 9 features for our detection.

4.1.5 Permissions

We use the the following 20 permissions as features in our detection. The first six permissions are derived from our empirical analysis of bad apps, which shows that many reported apps ask for credentials, contact list and hardware control, such as camera, audio or video. The next six permissions, such as `INSTALL_SHORTCUT` and `WRITE_HISTORY_BOOKMARKS`, are relevant to adware behaviors. The rest of the permissions, such as `BILLING` and `SEND_SMS`, are relevant to malware behaviors.

- android.permission.USE_CREDENTIALS

- android.permission.READ_CONTACTS
- android.permission.RECORD_AUDIO
- android.permission.CAMERA
- android.permission.CAPTURE_VIDEO_OUTPUT
- android.permission.CAPTURE_SECURE_VIDEO_OUTPUT
- android.permission.ACCESS_FINE_LOCATION
- com.android.launcher.permission.INSTALL_SHORTCUT
- android.launcher.permission.INSTALL_SHORTCUT
- com.android.browser.permission.READ_HISTORY_BOOKMARKS
- com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
- android.permission.WRITE_SETTINGS
- android.permission.INTERNAL_SYSTEM_WINDOW
- android.permission.BILLING
- android.permission.SEND_SMS
- android.permission.CALL_PHONE
- android.permission.PROCESS_OUTGOING_CALLS
- android.permission.INSTALL_PACKAGES
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.WRITE_EXTERNAL_STORAGE

4.1.6 API Calls

We identify 15 API calls that are required to complete specific behaviors of bad apps. Table 2 shows the list of API calls used in our detection. The first six API calls are required for adware behaviors, such as sending ads as notifications and changing browser settings. The rest of the APIs are used for Java reflection and dynamic code loading, since they are normally used by malicious apps to avoid being detected in static analysis. We extract 3 features from each API calls: presence of identified API calls, number of calls and whether the class files making the API calls are obfuscated. In total, we extract 45 features from API calls.

4.1.7 Others

There are other 4 types of features that we use for our detection. The first feature is whether the apps import cryptographic package `javax.crypto` because many malicious apps are known to encrypt and decrypt their codes. Another feature is whether the application files overwrite the `onBackPressed()` method, since this API is called by apps with exit ads. We also determine whether the APK codes include suspicious strings, such as `com.android.launcher.action.INSTALL_SHORTCUT`, since homescreen shortcuts may be added via intents with the above action strings. The final feature is whether apps include any string literals ending with ".apk", because many bad apps force users to download and install external APK files. Similar to API calls, we derive 3 types of information for each feature: presence of identified API calls, number of calls and whether the class files making the API calls are obfuscated

4.2 Detection

We apply 10 commonly used machine learning classifiers using Weka library [13]. The classifiers include Naive Bayes, Logistic, Sequential Minimal Optimization (SMO), Lazy-Ibk, Random Committee, Decision Table, Decision Part, J48, Logistic Model Tree (LMT), and Random Forest Tree. Naive Bayes is a family of simple probabilistic classifiers based on the Bayes' theorem. Logistic classifier applies the regression model, SMO applies Support Vector Machines (SVM), and Lazy - Ibk classifier applies K-nearest neighbours algorithm. Random Committee classifier uses a group of base classifiers, and the result is the average of the predictions generated by the individual base classifiers. Decision Table classifier uses a simple decision table. J48 Tree classifier, LMT classifier and Random Forest Tree classifier are algorithms based on decision trees.

5 Evaluation

In our evaluation, we apply the 10 classifiers to 628 apps used for our evaluation, including 302 reported bad apps and 326 benign apps. The 3-fold cross-validation is used for reducing over-fitting in our evaluation, where two-third of our data set is used for training and one-third is used for testing. Table 3 shows the weighted average of true positives, false positives, precision, recall and f-measures of our detection algorithms. True positive rates indicate the percentage of real malicious bad apps among the reported bad apps. False positive rates indicate the percentage of bad apps that are reported but are not really malicious. Precision is the ratio of true positives to true positives plus false positives. Recall is the ratio of true positives to true positives plus false negatives. F-measure combines precision and recall, and can be used as an overall measure for evaluation. For true positive rate, precision, recall and f-measure, the higher the scores are, the better the algorithm performs. The reverse is true for false positive rate.

In terms of f-measure, SMO classifier performs the best, followed by Random Committee and LMT classifiers. SMO classifier uses SVM, which is known to be the best classifier in many general cases. At the same time, Random Committee and LMT classifiers are good at dealing with binary and multi-class target variables, numeric and nominal attributes as well as missing values. Thus, they seem to be well-suited for our data set. The least effective classifiers are Naive Bayes, Logistic regression, and Decision Table classifiers. This is mainly due to the fact that Naive Bayes and Logistic regression classifiers perform the best for categorical

Algorithm	True Positive	False Positive	Precision	Recall	F-Measure
Naive Bayes	0.780	0.216	0.785	0.780	0.780
Logistic regression	0.774	0.228	0.774	0.774	0.774
SMO	0.868	0.136	0.871	0.868	0.867
Lazy-Ibk	0.828	0.177	0.833	0.828	0.827
Random Committee	0.855	0.148	0.857	0.855	0.855
Decision Table	0.750	0.256	0.754	0.750	0.748
PART	0.812	0.190	0.812	0.812	0.812
J48	0.815	0.188	0.817	0.815	0.815
LMT	0.854	0.151	0.857	0.854	0.853
Random Forest Tree	0.852	0.152	0.855	0.852	0.851

Table 3: Evaluation of Our Detection Algorithm

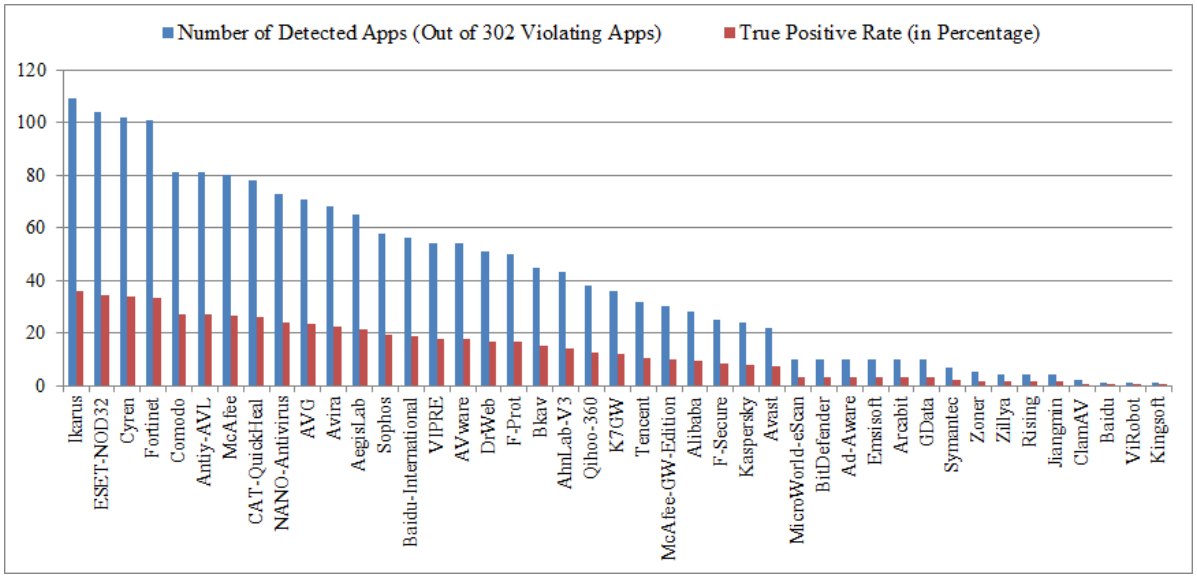

Figure 1: Detection Result by Anti-Virus Software from VirusTotal

dependent variables, while our dependent variables include non-categorical features, such as star ratings and numbers of downloads. Moreover, simple decision table classifier may not capture the complex rules of our feature set. We expect our results to be improved by focusing on the individual types of policy-violating apps. By doing so, we can select the features based on the behaviors specific to the types of apps. For instance, we can apply text and image similarity features for detecting intellectual property right violating apps. We leave this as future work, since the purpose of this paper is to characterize and detect all policy-violating apps.

We also compare our result with VirusTotal, which scans the uploaded apps with 57 anti-virus software. We submitted our 302 policy-violating apps to VirusTotal an retrieved the scanned report. Overall scan report shows that VirusTotal can detect only 168 of the submitted apps with its 57 anti-virus software. The remaining 134 apps are never alerted by any of the anti-virus software. At the same time, VirusTotal falsely reports 57 benign apps as policy-violating apps. Thus, we can say that the true positive rate of VirusTotal is 55.63% and the false positive rate of VirusTotal is 17.48%. The numbers of reported policy-violating apps (i.e. true positives) by individual anti-virus solutions are shown in Figure 1.

6 Related Work

Although there are limited studies on intellectual property right violating apps, there are many studies on the repackaged apps. Repackaged apps are the clones created from the reverse-engineered codes of original apps. Balanza et al.[2] analyze a repackaged malware, called DroidDreamLight, and Jung et al. [17] launch repackaging attack on bank apps. Chen et al. [4], and Gibler et al. [10] study the

impact of repackaged apps and find out that 14% of original developers' revenues and 10% of user are redirected to the attacker. Potharaju et al. [22] use permission information and estimate that 29.4% of apps are likely to be plagiarized.

Since repackaged apps contain similar source codes as original apps, their detection mechanisms focus on the source code similarities. DroidMOSS [33] and Juxtapp [14] [18] apply fuzzy hashing on program instruction sequence and derive the similarity score by calculating the edit distance between two generated fingerprints. Crussell et al. [6] propose DNADroid, which uses Program Dependence Graph(PDG) to determine code similarity. AnDarwin [7] applies Locality-Sensitive Hashing (LSH), Lin et al. [19] use thread-grained system call sequences and Zhou et al. [32] propose linearithmic search algorithm in a metric space to detect repackaged apps. Deckard [16] uses a tree-based detection algorithm for detection. Huang et al. [15] propose an evaluation framework for detection algorithms of repackaged apps by measuring their resilience to obfuscation methods. Different from other approaches, Zhou et al. [31] propose to use software watermarking to prevent repackaging. Since these methods only focuses on code similarity, they cannot detect apps, which copy the external features of original apps and not their source codes.

Similar to our paper, several previous work highlights various issues of ad libraries. However, they focus more on privacy, security and usability issues, and not on their aggressiveness for showing ads or obtaining clicks from users. Adrisk [12] applies static analysis on top 100 commonly used ad libraries, and shows that most ad libraries collect private information, including users' location, call logs, phone number, browser bookmarks, and the list of installed apps on the phone. Moreover, some libraries directly fetch and run code from the Internet. Book et al. [3] make a longitudinal analysis of permissions used by ad libraries, and discovers that dangerous permission usage by ad libraries are increasing over time.

7 Conclusion

In this paper, we perform extensive empirical analysis on bad apps that are reported and removed from Google Play store. These bad apps are diligently collected by crawling Reddit forum posts and Google Play store over a three-month period. Our analysis of the data set provides a comprehensive overview of reported bad apps and their policy-violating behaviors. Our findings show that detecting copyright violating apps and ad-aggressive apps is important for maintaining good quality of future mobile app market. Thus, we urge industry and research communities to give more attention to these areas. Our paper also includes detection of bad apps using machine learning classifiers. We derive features based on the results of our findings as well as behavior-based features. Although our current solution is performing well for detecting policy-violating apps, we believe that better solutions can be invented by focusing on each type of policy-violating apps.

References

[1] Y. Aafer, W. Du, and H. Yin. *Security and Privacy in Communication Networks: 9th International ICST Conference, SecureComm 2013, Sydney, NSW, Australia, September 25-28, 2013, Revised Selected Papers*, chapter DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android, pages 86–103. Springer International Publishing, Cham, 2013.

[2] M. Balanza, O. Abendan, K. Alintanahin, J. Dizon, and B. Caraig. Droiddreamlight lurks behind legitimate android apps. In *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software*, MALWARE '11, pages 73–78, Washington, DC, USA, 2011. IEEE Computer Society.

[3] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal analysis of android ad library permissions. *CoRR*, abs/1303.0857, 2013.

[4] H. Chen. Underground economy of android application plagiarism. In *Proceedings of the First International Workshop on Security in Embedded Systems and Smartphones*, SESP '13, pages 1–2, New York, NY, USA, 2013. ACM.

[5] N. Chen, S. C. Hoi, S. Li, and X. Xiao. Simapp: A framework for detecting similar mobile applications by online kernel learning. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 305–314, New York, NY, USA, 2015. ACM.

[6] J. Crussell, C. Gibler, and H. Chen. Attack of the clones: Detecting cloned applications on android markets. In *Proceedings of 17th European Symposium on Research in Computer Security*, pages 37–54, 2012.

[7] J. Crussell, C. Gibler, and H. Chen. Scalable semantics-based detection of similar android applications. In *18th European Symposium on Research in Computer Security*, ESORICS '13, Egham, U.K., 2013.

[8] A. Desnos. Androguard, 2011.

[9] C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi. Adrob: Examining the landscape and impact of android application plagiarism. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 431–444, New York, NY, USA, 2013. ACM.

[10] C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi. Adrob: Examining the landscape and impact of android application plagiarism. *Proceedings of 11th International Conference on Mobile Systems, Applications and Services*, 2013.

[11] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 281–294, New York, NY, USA, 2012. ACM.

[12] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 101–112, New York, NY, USA, 2012. ACM.

[13] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[14] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song. Juxtapp: a scalable system for detecting code reuse among android applications. In *Proceedings of the 9th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'12, pages 62–81, Berlin, Heidelberg, 2013. Springer-Verlag.

[15] H. Huang, S. Zhu, P. Liu, and D. Wu. A framework for evaluating mobile app repackaging detection algorithms. In M. Huth, N. Asokan, S. apkun, I. Flechais, and L. Coles-Kemp, editors, *Trust and Trustworthy Computing*, volume 7904 of *Lecture Notes in Computer Science*, pages 169–186. Springer Berlin Heidelberg, 2013.

[16] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 96–105, Washington, DC, USA, 2007. IEEE Computer Society.

[17] J.-H. Jung, J. Y. Kim, H.-C. Lee, and J. H. Yi. Repackaging attack on android banking applications and its countermeasures. *Wirel. Pers. Commun.*, 73(4):1421–1437, Dec. 2013.

[18] S. Li. Juxtapp and dstruct: Detection of similarity among android applications. Master's thesis, EECS Department, University of California, Berkeley, May 2012.

[19] Y.-D. Lin, Y.-C. Lai, C.-H. Chen, and H.-C. Tsai. *Identifying android malicious repackaged applications by thread-grained system call sequences*. Computers and Security, Greenwich, CT, USA, 2013.

[20] C. Mann and A. Starostin. A framework for static detection of privacy leaks in android applications. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1457–1462, New York, NY, USA, 2012. ACM.

[21] N. Peiravian and X. Zhu. Machine learning for android malware detection using permission and api calls. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, ICTAI '13, pages 300–305, Washington, DC, USA, 2013. IEEE Computer Society.

[22] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang. Plagiarizing smartphone applications: attack strategies and defense techniques. In *Proceedings of the 4th international conference on Engineering Secure Software and Systems*, ESSoS'12, pages 106–120, Berlin, Heidelberg, 2012. Springer-Verlag.

[23] C. Soh, H. B. K. Tan, Y. L. Arnatovich, and L. Wang. Detecting clones in android applications through analyzing user interfaces. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, ICPC '15, pages 163–173, Piscataway, NJ, USA, 2015. IEEE Press.

[24] O. Tripp and J. Rubin. A bayesian approach to privacy enforcement in smartphones. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 175–190, San Diego, CA, Aug. 2014. USENIX Association.

[25] VirusTotal. Virustotal public api v2.0.

[26] N. Xia, H. H. Song, Y. Liao, M. Iliofotou, A. Nucci, Z.-L. Zhang, and A. Kuzmanovic. Mosaic: Quantifying privacy leakage in mobile networks. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 279–290, New York, NY, USA, 2013. ACM.

[27] Z. Yang and M. Yang. Leakminer: Detect information leakage on android with static taint analysis. In *Proceedings of the 2012 Third World Congress on Software Engineering*, WCSE '12, pages 101–104, Washington, DC, USA, 2012. IEEE Computer Society.

[28] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang. Appintent: analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13, pages 1043–1054, New York, NY, USA, 2013. ACM.

[29] Y. Zhauniarovich, O. Gadyatskaya, B. Crispo, F. La Spina, and E. Moser. *Data and Applications Security and Privacy XXVIII: 28th Annual IFIP WG 11.3 Working Conference, DBSec 2014, Vienna, Austria, July 14-16, 2014. Proceedings*, chapter FSquaDRA: Fast Detection of Repackaged Applications, pages 130–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[30] W. Zhou, Z. Wang, Y. Zhou, and X. Jiang. Divilar: Diversifying intermediate language for anti-repackaging on android platform. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, pages 199–210, New York, NY, USA, 2014. ACM.

[31] W. Zhou, X. Zhang, and X. Jiang. Appink: Watermarking android apps for repackaging deterrence. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 1–12, New York, NY, USA, 2013. ACM.

[32] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou. Fast, scalable detection of "piggybacked" mobile applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, CODASPY '13, pages 185–196, New York, NY, USA, 2013. ACM.

[33] 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*, CODASPY '12, pages 317–326, New York, NY, USA, 2012. ACM.

[34] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.

[35] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings*, chapter Taming Information-Stealing Smartphone Applications (on Android), pages 93–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.