Hindawi
Mobile Information Systems
Volume 2017, Article ID 7397812, 16 pages
https://doi.org/10.1155/2017/7397812

Research Article

Selecting Critical Data Flows in Android Applications for Abnormal Behavior Detection

Pengbin Feng,[1,2] Jianfeng Ma,[1,2] and Cong Sun[1]

[1]*School of Cyber Engineering, Xidian University, Xi'an, China*
[2]*School of Computer Science and Technology, Xidian University, Xi'an, China*

Correspondence should be addressed to Pengbin Feng; pbfeng@outlook.com

Received 9 December 2016; Revised 22 March 2017; Accepted 4 April 2017; Published 30 April 2017

Academic Editor: Maristella Matera

Nowadays, mobile devices are widely used to store and process user privacy and confidential data. With the popularity of Android platform, the cases of attacks against users' privacy-sensitive data within Android applications are on the rise. Researchers have developed sophisticated static and dynamic analysis tools to detect information leakage. These methods cannot distinguish legitimate usage of sensitive data in benign apps from the intentional sensitive data leakages in malicious apps. Recently, malicious apps have been found to treat sensitive data differently from benign apps. These differences can be used to flag malicious apps based on their abnormal data flows. In this paper, we further find that some sensitive data flows show great difference between benign apps and malware. We can use these differences to select critical data flows. These critical flows can guide the identification of malware based on the abnormal usage of sensitive data. We present SCDFLOW, a tool that automatically selects critical data flows within Android applications and takes these critical flows as feature for abnormal behavior detection. Compared with MUDFLOW, SCDFLOW increases the true positive rate of malware detection by 5.73%~9.07% on different datasets and causes an ignorable effect on memory consumption.

1. Introduction

According to a recent study [1], Android has been growing market share in the smartphone market, which in the second quarter of 2016 is at 87.6%. With Android phones being ubiquitous, they unsurprisingly become worthwhile targets for malicious attacks. McAfee reports [2] that almost 6 million new mobile malware instances were collected in 2015, which indicates a dramatic increase over 2014. The number of Android malware samples has been growing explosively over the past few years. Android platform allows for installing applications from uncertified application provider, which makes Android vulnerable to malware. It is evident that there is an urgent need for effective and precise malware detection to stop the proliferation of malware targeting Android platform.

A large amount of research has proposed methods for analyzing and detecting Android malware. These methods can be roughly categorized into methods using static and dynamic analysis. For instance, DroidRanger [3], AppsPlayground [4], and CopperDroid [5] effectively identify malware through monitoring the behaviors of an application at runtime but suffer from significant overhead. Static analysis methods, such as Kirin [6], Stowaway [7], and RiskRanker [8], check applications' behavior features like requested permissions and suspicious API calls to detect malware. However, these methods are mainly built on manually crafted detection rules which are often hard to update with the explosively growth of malware instances.

Machine learning-based approaches can automatically infer detection patterns from features extracted by program analysis from malware and have become promising approaches for detecting malware. Drebin [9], DroidDolphin [10], and MobileSandbox [11] perform static or dynamic analysis or a combination of both analyses to extract features (such as requested permissions and API calls) from Android applications and adopt machine learning algorithms to perform malware classification. These methods train the classifier on samples of known malware and benign apps, which

are effective at detecting other samples of similar malware but quite ineffective at detecting new types of malware which are sufficiently different from known malware.

Although Android malware has become more and more sophisticated over the past few years, privacy violations that leak sensitive information have always been one of the main threats posed by malicious Android applications. In this paper, we focus on information leakages behaviors of Android applications and aim at detecting malware causing sensitive information leakages. Many research efforts have developed taint analysis techniques to detect information leakages. However, the sole rule that an app leaks specific sensitive information does not indicate that the app is malicious. Malicious applications usually leak sensitive information such as location information, contact data, and SMS messages to attackers. Nevertheless, even benign apps may need to legitimately access sensitive information; for instance, an application needs to use sensitive account information to manage synchronization across multiple devices. Therefore, these taint analysis approaches alone cannot precisely distinguish malware from benign applications.

In this work, we apply a one-class classifier that trains a classification model on sensitive data flows of benign apps. With this classifier, we can identify new types of malware based on its abnormal data flows even if no earlier similar malware samples are known. Avdiienko et al. [12] have found that sensitive data flows within malicious apps are considerably different from the ones within benign apps. These differences can be used to flag malicious apps based on their abnormal data flows. MUDFLOW [12] applies static taint analysis on a representative set of benign apps and then obtains a data-flow set consisting of "normal" usage of sensitive data. It then measures the distances between the sensitive data usage behaviors of different apps and takes these distances as features to train a classification model. When a novel malware comes, MUDFLOW would identify it as potential malware based on its abnormal usage of sensitive data.

MUDFLOW points out that malicious apps treat sensitive data differently from benign apps. In this work, we go further to show that some sensitive data flows frequently emerge in both benign apps and malware. For example, Android applications usually record network and database operation in LOG. Consequently, data flows like *TelephonyManager.getNetworkOperatorName*() $\rightsquigarrow$ *Log.d*() and *SQLiteDatabase.query*() $\rightsquigarrow$ *Log.i*() frequently emerge in both benign apps and malware. These data flows are irrelevant and noisy when distinguishing malware from benign apps. Taking these data flows as features will cause negative effect on malware identification. Removing these irrelevant data flows can improve the precision of malware detection based on abnormal usage of sensitive data.

In this paper, we also find that occurrence frequency of sensitive data flows shows great difference between benign apps and malware. For example, malicious applications usually send subscriber ID and device ID through SMS. However, data flows like *TelephonyManager.getSubscriberId*() $\rightsquigarrow$ *SmsManager.sendTextMessage*() and *TelephonyManager.getDeviceId*() $\rightsquigarrow$ *SmsManager.sendTextMessage*() hardly emerge within benign apps. The difference on occurrence frequency of each data flow is able to derive critical data flows. Measuring difference on these flows can increase the distinction between malware and benign apps. We select critical data flows according to the difference on occurrence frequency of each flow in benign apps and malware. Taking only these flows as features improves the precision of malware detection based on abnormal data flows. We make the following contributions:

(1) We present SCDFLOW (Selecting Critical Data Flows), an approach that selects critical sensitive data flows and then takes these flows as features to report potential novel malware.

(2) We propose a novel feature selection algorithm *CFlowSel*, which selects critical data flows based on frequency difference of each flow. These critical flows can guide the identification of malware based on abnormal usage of sensitive data. Through experiments, we verify that *CFlowSel* outperforms existing feature selection algorithms.

(3) On datasets published by MUDFLOW and DREBIN, we verify that this approach can improve the classification precision when compared with MUDFLOW. The malware detection rates are, respectively, increased by 5.73% on MUDFLOW dataset and 9.07% on DREBIN dataset.

The rest of this paper is organized as follows. Section 2 explains the necessary background of outlier detection and information flow analysis for Android applications. Section 3 discusses the implementation details of our abnormal detection framework and presents our feature selection algorithm. Section 4 contains experiment settings and datasets used in our experiments and presents the measure metrics used to evaluate the classification performance. Section 5 reports the evaluation of our experimental results. Section 6 presents malware case studies to illustrate the effectiveness of our method. Section 7 discusses the threats to validity of our approach. Section 8 discusses the relation to existing work. Section 9 concludes this paper and presents future work.

2. Background and Preliminaries

In this section, we briefly describe the methodology of outlier detection, the technique of information flow analysis, and the definition of sensitive data sources and sinks in Android applications.

2.1. Outlier Detection. Outlier detection techniques have been well studied in the data mining, machine learning, and statistics literature. Outlier detection is aimed at solving the problem that finds patterns that do not conform to a well-defined notion of normal behavior in given data. The anomalous patterns often refer to exceptions, faults, defects or errors, and so forth.

Distance-based outlier detection is a popular approach to finding anomalous examples in a dataset. This approach is based on distance measures on feature spaces and capable of processing data with high dimensionality. It is able to assign

an outlier score to an example based on the degree to which this example is considered abnormal. A popular method of unusualness representation is by examining the distance to an example's nearest neighbors. In this approach, one looks at the local neighborhood of points for an example which is typically defined by k nearest examples [13]. An example is considered normal if its neighboring points are relatively close; meanwhile, this example is considered unusual if its neighboring points are far away. Distance-based detections own two advantages that they do not need to define explicit distribution to determine unusualness and that they can be applied to any feature space for which we can define a distance measure.

There are usually a large number of features in outlier detection problems. Some of these features are noisy or irrelevant and cause problems such as degrading the performance of outlier detection and increasing model complexity and detection time. Feature selection, the method of selecting most relevant features for building appropriate detection models, is able to remove useless features. This method can also speed up a classification task and improve classification performance such as prediction accuracy and result comprehensibility.

2.2. Information Flow Analysis. When installing an Android application, users can only see a textual description of the function that the app alleges and a list of permissions that the app requires. An app may require accessing the device ID of the Android smartphone. However, it is unclearly stated in the description how the app deals with this data. Wang et al. [14] state that the behavior of Android application cannot be informatively described by the list of request permissions. Therefore, in this paper, we use sensitive data flows to characterize the behaviors of Android application, which indicate the usage of sensitive data. In essence, given a source of sensitive data (e.g., the location information containing the longitude and latitude) and a sink (an Http connection to a third-party server), this kind of flow is defined as the information leakage identified from the source to the sink. Box 1 shows an example of how an Android application leaks location information. The example reads the location information and saves it to text file "privacyfile.txt". In real-world applications, the flow from source (the call to *getLastKnowLocation*()) to sink (the method call to *stringToFile*()) can be more complex, may involve multiple components, and may include field accesses, conditionals, and polymorphic method calls.

Researchers have used information flow analysis to identify whether confidential information of sources can flow to undesired sinks. Both static and dynamic information flow analysis have been used to detect information leakage within Android apps. Static information flow analysis would report all possible existing data flows in the program from sensitive sources to undesired sinks. This analysis constructs control flow graph and function call graph of application, seeks sensitive data, and checks whether sensitive data flows to undesired sink. Meanwhile, static analysis might report false positives, that is, reporting data flows that are not feasible in practice. Dynamic information flow analysis monitors the execution of application and real-timely traces the propagation of sensitive data. This analysis would report sensitive data flows that actually occur during a specific test but miss flows that are unreachable in this test.

Internally, SCDFLOW leverages highly precise static analysis tool FLOWDROID [15] to extract sensitive data flows within Android applications. It precisely models the lifecycle of Android applications and properly handles callbacks invoked by Android framework. Android applications comprise many entry points instead of a main method and are coupled with their application framework that can start, stop, pause, or resume them depending on environmental state. FLOWDROID first analyzes the applications for all registered components and event handler and then creates dummy code that simulates interactions with operating system. FLOWDROID provides a fully object-, flow-, and context-sensitive highly precise taint analysis. This kind of highly precise taint analysis can not only reduce the false positives during data flow analysis but also reduce the amount of noise within the input data on selecting critical sensitive data flows. For example, FLOWDROID is able to identify information leakage in Box 1 from *getLastKnowLocation*()(LOCATION) to *StringToFile*()(FILE) in Box 1.

2.3. Sensitive Sources and Sinks. The identification of sensitive information leakages requires the definition of sensitive sources and sinks in Android applications. Sensitive information such as location information, contact data, pictures, and SMS messages can only be accessed through specific API methods. For instance, the method *getLastKnowLocation*() returns the location information of Android smartphone. For tracing the flow of sensitive information, API methods that access and leak sensitive resources need to be identified. Many research efforts have given a definition of sources and sinks in Android applications. These methods only manually identify a set of sources and sinks, which contain a small fraction of well-known Android API methods. Furthermore, manually crafted lists get outdated with every new Android version.

In this paper, we leverage SUSI [16] to automatically identify sensitive sources and sinks in Android API methods. This approach identifies both semantic and syntactic features of source and sink methods and trains a classification model from a small hand-annotated fraction of Android APIs. This classification model divides the whole API methods into source, sink, or neither. To provide a list of APIs that access sensitive sources and leak to sensitive sinks, SUSI further provides a further categorization of these sources and sinks. For instance, the API method *getLastKnowLocation*() belongs to source category LOCATION_INFORMATION and API method *StringToFile*() belongs to sink category FILE.

3. Approach Overview

In this section, we first introduce the overall architecture of SCDFLOW. Next, each module is described individually to explain how SCDFLOW works for abnormal behavior detection. Then, we detail our methodology for selecting critical data flows within Android apps.

```
(1) void onCreate(){
(2)       locationManager = (LocationManager)
              getSystemService(Context.LOCATION_SERVICE);
(3)       Location location = locationManager
              .getLastKnownLocation(LocationManager.GPS_PROVIDER);
(4)       String content = "Longitude:" + location.getLongitude()
              + "Latitude:" + location.getLatitude();
(5)       String file = this.getFilesDir() + "/privacyfile.txt";
(6)       FileUtils.stringToFile(file,content);
(7) }
```

Box 1: Android location leak example.

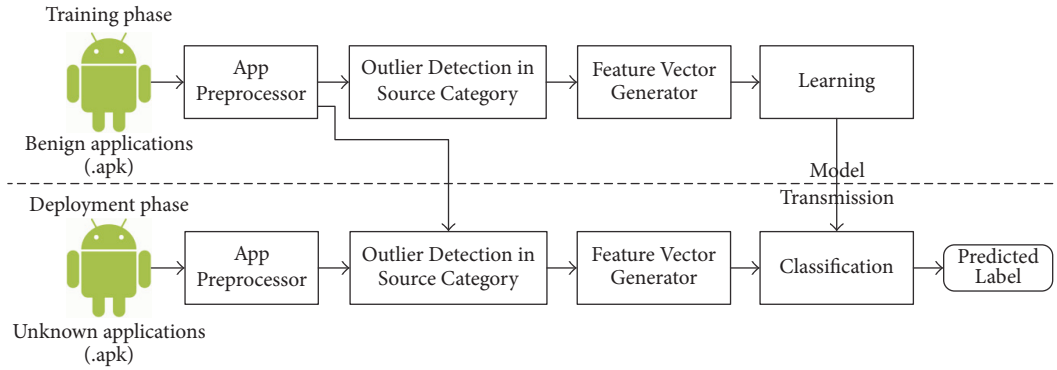


Figure 1: The overall framework of SCDFLOW.

3.1. Architecture of SCDFLOW. As shown in Figure 1, our framework contains two phases: training phase and deployment phase. For the training phase, the framework takes package files of benign applications as input and outputs a classification model that is able to identify malware based on its abnormal data flows. This learned classification model would then be passed to the deployment phase. For the deployment phase, given an unknown application, the framework preprocesses its package files and applies the classification model generated from the training phase to decide whether this application is a malicious app or a benign app.

3.1.1. Training Phase. In the training phase, SCDFLOW has four components: *App Preprocessor*, *Outlier Detection in Source Category*, *Feature Vector Generator*, and *Learning*. First, *App Preprocessor* performs static taint analysis to extract all sensitive data flows within applications. Next, *Outlier Detection in Source Category* calculates outlier scores for every app in each source category. Then, *Feature Vector Generator* generates feature vectors for every app by combining its outlier scores across all source categories. Finally, *Learning* component trains a classifier by taking the feature vectors of benign apps as inputs. Here, we choose a one-class classifier to generate the classification model for reporting potential novel malware instance.

3.1.2. Deployment Phase. In the deployment phase, SCDFLOW also has four components: *App Preprocessor*, *Outlier Detection in Source Category*, *Feature Vector Generator*, and *Classification*. SCDFLOW applies the same preprocessor, outlier detection, and feature vector generator steps on unknown apps. Then, the *Classification* component uses the classification model generated from the training phase to process these feature vectors of unknown apps and predicts which apps are malware or benign.

3.2. App Preprocessor. The behaviors of an Android application are described by the usage of sensitive source. For a single Android application a, SCDFLOW uses FLOWDROID to extract all data flows from sensitive sources to sensitive sinks. The result of this information flow analysis is a set of API method pairs that characterize the usage of sensitive data within the application and the result has the form

$$DataFlows(a) = \{se_1 \rightsquigarrow sk_1, se_2 \rightsquigarrow sk_2, \ldots\}, \quad (1)$$

where se denotes one sensitive source and sk denotes one sensitive sink. $\{se \rightsquigarrow sk\}$ denotes one sensitive data flow from se to sk.

As input for *Outlier Detection in Source Category* component, information leakage behaviors of Android applications must be represented by data-flow vectors. Let F denote all sensitive data flows within all Android applications. Given a, for each sensitive data flow $f_i \in F$, we value it as 1 for the cases that $f_i \in DataFlows(a)$ or as 0 for the cases that $f_i \notin DataFlows(a)$. Let a_i denote a containing sensitive data flow

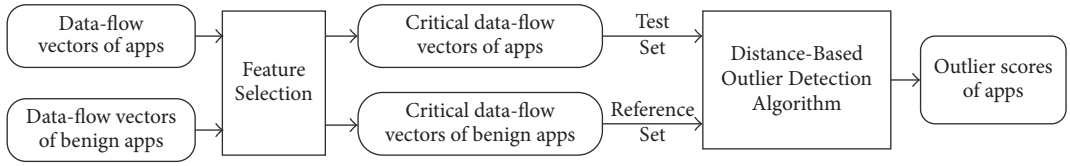


Figure 2: The workflow of *Outlier Detection in Source Category*.

f_i; the data-flow vector of a has the form $FlowVector(a) = \{a_1, a_2, \ldots, a_i, \ldots\}$.

3.3. Outlier Detection in Source Category. The workflow of *Outlier Detection in Source Category* component is shown in Figure 2. First, this component uses *Feature Selection* algorithm to select critical sensitive data flows. Based on these critical data flows, this component can reduce the dimensionality of data-flow vectors and generate critical data-flow vectors for each app. Finally, considering these critical sensitive data flows as feature, the outlier detection algorithm examines each example in test set and determines its outlier score by computing average distance to its k-nearest neighbors in reference set.

In training phase, the test set is the critical data-flow vectors of benign apps. However, in deployment phase, it is the critical data-flow vectors of unknown apps. For simplicity, we consider critical data-flow vectors of both benign apps and unknown apps as test set. Let s denote one SUSI category of sensitive sources. Let B denote a set of benign apps and C denote a set of unknown apps. Apps in B and C that use at least one API from s are, respectively, represented by B_s and C_s. Specifically, for category s, SCDFLOW extracts all sensitive data flows within B_s and C_s, takes B_s from app set B and C_s from app set C, and finally denotes sensitive data flows within B_s and C_s as data-flow vectors.

The *Outlier Detection in Source Category* component works in two steps, detailed below.

3.3.1. Feature Selection. Taking data-flow vectors of B_s and C_s as well as necessary parameters, feature selection algorithm (detailed in Section 3.7) outputs a set of critical sensitive data flows. Most data-flow features are worthless for classification purpose, and only small proportion of them are enough to capture the distinguished characteristics between benign apps and malware. These features represent data flows that are informative and critical for distinguishing malware from benign apps. Based on these critical flows, the data-flow vectors of each app are transformed into critical data-flow vectors.

3.3.2. Distance-Based Outlier Detection Algorithm. Taking critical data-flow vectors of B_s and C_s as test set and critical data-flow vectors of B_s as reference set, outlier detection algorithm ORCA [13] is used to determine outlierness of each app in test set. In this work, we used *weighted Jaccard* distance metric to measure the dissimilarity between applications since it is more suitable for data with a huge number of features. By default, the outlierness of each app is represented by the average distance to its 5 nearest neighbors in reference set. The resulting outlierness is represented by an outlier score. The higher the outlier score, the less "normal" its data-flow features, resulting in a higher likelihood of being malicious.

3.4. Feature Vector Generator. *Outlier Detection in Source Category* outputs the outlier scores for each application with respect to "normal" sensitive data flows. *Feature Vector Generator* component applies the above outlier detection on all source categories, resulting in a single outlier detector for each sensitive source category. For a single app a, this component determines its outlier scores across all categories. These scores tell for each category how much a deviates from the norm. If an application uses no API method from appropriate source category, its outlier score in that category is set to zero. Feature vector of an application is derived by aggregating outlier scores on each category. Suppose $score_i$ denotes the outlier score of a in source category s_i, a's feature vector can be expressed as $FeatureVector(a) = \{score_1, score_2, \ldots, score_i, \ldots\}$.

3.5. Learning. The *Learning* component takes the feature vectors of benign apps as input and trains a classification model based on v-SVM one-class classifier [17]. Therefore, this component could learn to classify feature vectors that correspond to benign apps as normal and those that correspond to malicious apps as abnormal.

The resulting classification model can be used for detecting novel and unknown elements. The one-class SVM classifier leverages kernel function to project the training data into a high dimensional feature space and iteratively seeks the maximal margin hyperplane that best separates the training data from the origin. Then, this classifier can be treated as standard two-class SVM where all training data is taken as the first class, and the origin is taken as the only member which lies in the second class. Eventually, the test data is classified by projecting it into that high dimensional space and checking which side of the hyperplane it falls.

3.6. Classification. Given unknown apps, we use the same components *App Preprocessor*, *Outlier Detection in Source Category*, and *Feature Vector Generator* as training phase to generate feature vectors. Based on these vectors, the *Classification* component could leverage the classification model to decide which apps are malicious or which apps are benign and outputs their category labels.

3.7. Feature Selection. We assume the most popular Android apps from Google Play Store to be benign and consider sensitive data flows within these apps to constitute the "normal" usage of sensitive data. Unknown apps contain malicious apps as well as less popular apps obtained from third-party Android markets. Third-party Android markets cannot perform sufficient security checking. Therefore, less popular apps from these markets may contain malicious behavior. In feature selection, we take all benign and unknown apps as input and consider unknown app as malware to select critical sensitive data flows.

With *App Preprocessor*, we obtain 3733 sensitive data flows in all. Malicious apps have been found to treat sensitive data differently from benign apps. Therefore, sensitive data flows within malicious apps differ significantly from the ones within benign apps. However, some data flows (e.g., *Cursor.query*() $\rightsquigarrow$ *AlertService.generateAlerts*()) are only used by very few apps in our dataset, and some other flows (e.g., *SQLiteDatabase.getWritableDatabse*() $\rightsquigarrow$ *Log.e*()) are widely used by both malware and benign apps. These data flows contribute less to distinguishing malware from benign apps. Moreover, taking these flows as features may cause complicated computation and low efficiency in building classification. Therefore, we take feature selection procedure before abnormal behavior detection.

Feature selection methods are aimed at selecting a subset of features that can efficiently describe the training data while reducing effects from noise or irrelevant features and still providing good prediction results. In two-class malware detection, feature selection aims to select most informative features that malware often manipulates. The feature selection methods essentially divide into wrapper, filter, and embedded methods. Embedded methods perform feature selection in the process of training and are usually specific to given classification algorithm. In wrapper methods, the criterion of feature selection is the performance of a classifier and the classifier is wrapped on a searching algorithm that would find a subset that achieves the highest classification performance. Filter methods act as preprocessing to rank the features wherein the highly ranked features are selected and applied to training a classifier. Embedded methods, which are usually specific machine learning algorithms, cannot be applied to our abnormal detection framework. The time complexity of wrapper methods would grow exponentially for higher number of features. With comprehensive consideration of time and performance, we choose filter methods.

In the following, we describe two well-known filter methods, *mutual information* and *chi-square*, highlight the drawbacks of these two methods in our problem settings, and propose a novel feature selection algorithm *CFlowSel*. *CFlowSel* selects critical data flows based on occurrence frequencies of each data flow among datasets.

Mutual information is a measure of the mutual dependence between the two variables. Let D denote a random variable indicating the class of application, *malicious* or *benign*. Every application is assigned a data-flow vector $(x_1, x_2, \ldots, x_i, \ldots)$ with x_i being the value of the ith sensitive data flow f_i. The relevance of f_i and D measured by mutual information can be represented by

$$MI(f_i, D) = \sum_{x_i \in \{0,1\}} \sum_{d_j \in \{ben, mal\}} P\left(f_i = x_i, D = d_j\right) \times \log \frac{P\left(f_i = x_i, D = d_j\right)}{P\left(f_i = x_i\right) P\left(D = d_j\right)}, \quad (2)$$

where $P(D = d_j)$ denotes the frequency count of class D with value d_j, $P(X_i = x_j)$ denotes the frequency count of sensitive data flow f_i with value x_j, and $P(f_i = x_j, D = d_j)$ denotes the frequency count of f_i with the value x_i in class d_j. $MI(f_i, D)$ is nonnegative in $[0, 1]$. $MI(f_i, D) = 0$ indicates that D has no correlation with sensitive data flow f_i, while $MI(f_i, D) = 1$ means that D can be completely inferable from sensitive data flow f_i.

Chi-square is a method to test the independence of two variables. In this paper, chi-square tests whether a sensitive data flow f_i is relevant to a class d. The chi-square value of χ^2 of sensitive data flow f_i and category d is shown as

$$\chi^2(f_i, d) = \frac{I \times (EG - FH)^2}{(E + G) \times (F + H) \times (E + F) \times (G + H)}, \quad (3)$$

where $I = E + F + G + H$, E and F are the count of benign and malicious apps that contain sensitive data flow f_i, and G and H are the count of benign and malicious apps that do not contain sensitive data flow f_i. The importance of f_i is directly proportional to $\chi^2(f_i)$.

For each sensitive data flow f_i, its relevance to class D can be evaluated by the mutual information and chi-square. We are especially interested in the sensitive data flows that are strongly correlated with D. Therefore, we sort sensitive data flows in decreasing order according to $|MI(f_i, d)|$ and $|\chi^2(f_i, d)|$ and select the top data flows for classification.

We find that 70.47% of the data-flow features get $\chi^2(f_i)$ less than 10.0 and 83.15% of the data-flow features obtain $MI(f_i)$ under 0.01. Therefore, most of data-flow features are unimportant, and they contribute little to machine learning algorithm. Moreover, these two algorithms only sort all data-flow features without indicating the count of features to be selected. We have to empirically set the count and select critical data flows. In addition, in this abnormal behavior detection framework, applications are grouped into different categories. In these categories, apps hold different behavior characteristics. The respective count of critical data flows in each category should be different. Therefore, it is hard to decide the count of data-flow features for every category.

Based on our experiments, compared with mutual information and chi-square, *CFlowSel* is able to select less data-flow features and achieves higher classification performance on the same count of data-flow features. After conducting a series of experiments to derive the maximum F-measure of mutual information, chi-square and *CFlowSel*, we find the maximum F-measure of *CFlowSel* is higher than the ones of mutual information and chi-square. In *CFlowSel*, we assume

N_m and N_b are the number of samples of malware M and benign apps B; then we examine this condition:

$$\left|\frac{T_m}{N_m} - \frac{T_b}{N_b}\right| > t, \tag{4}$$

where T_m is the number of samples of malware that contains a certain flow, T_b indicates the number of samples of benign apps contain the certain feature, and t denotes threshold of difference, $0 \leq t \leq 1$. This condition means that the data flow is used more frequently in malware than benign apps or more frequently in benign apps than malware. A data flow should be selected as a critical data flow once it satisfies this condition. In this way, critical data flows we collected not only increase the dissimilarity between benign apps and malware but also have a certain coverage in the feature dataset. In our abnormal behavior detection framework, feature selection aims at selecting most informative features that are more frequently used in malware than benign apps or more frequently used in benign apps than in malware. Based on these features, the difference between benign apps and malware maximizes. Existing filter methods pick features by calculating their importance. Correspondingly, *CFlowSel* selects features by finding the differences on frequency between malware and benign apps. In abnormal detection scene, features that are more frequently used in one category are more important to distinguish malware from benign apps.

In the following, we first provide the definition of the threshold-difference and then present the detailed *CFlowSel* (our algorithm implementation is available online at https://github.com/fpb1386/feature_selection) algorithm.

Definition 1. For each data flow, a *frequency difference* denotes the value of its occurrence frequency in malware minus the value of its occurrence frequency in benign apps.

Definition 2. Critical data flows denote the most informative flows that are sufficient to distinguish malware from benign apps.

Definition 3. Threshold-difference denotes the absolute minimum value of frequency difference of critical data flows.

In Algorithm 1, V_b denotes data-flow vectors of benign apps set B. V_m denotes data-flow vectors of malicious apps set M. t denotes threshold-difference. The method $countInSet(f, Y)$ calculates the total count of elements which contain sensitive data flow f in data-flow vector set Y. The method $card(Y)$ calculates the count of elements in data-flow vector set Y. In *CFlowSel*, data flows that appear more frequently in one category than in the other are selected. The main reason of this selection strategy is that we attempt to keep the balance of TPR and TNR (Section 4.2 details the definition of TPR and TNR). If we only use flows that are used frequently in malware, the TNR will be relative lower.

4. Experiments

This section reports the experimental setting used in SCDFLOW and the evaluation parameters.

4.1. Experiment Settings. SCDFLOW leverages static taint analysis tool FLOWDROID to extract sensitive data flows within Android applications. Aiming at providing highly precise analysis results, FLOWDROID uses a lot of time and resource to run taint analysis on real-world applications. In favor of a faster analysis and the ability to analyze a larger application, SCDFLOW takes the same FLOWDROID setting (no flow across intents, explicit flow only, flow-insensitive alias, maximum access path length of 3, no-layout mode, and no static files) as MUDFLOW chooses. This kind of analysis setting sacrifices some amount of precision for speed and memory. As a result, the list of data flows determined by FLOWDROID may have false positives (data flows are not feasible in practice) as well as false negatives (missing data flows that are possible in the actual runtime).

The machine we used to perform information flow analysis was a workstation with Intel(R) Xeon(R) E5-2620 CPU (15 M Cache, 2 GHz) and 24 GB of RAM. We set the time out for analyzing one single Android app to 12 hours. Meanwhile, we compare MUDFLOW with SCDFLOW on a PC, which is equipped with Inter(R) Core(TM) i7 CPU (6 M Cache, 2.5 GHZ) and 8 GB of RAM.

4.2. Evaluation Metrics. There have been proposed several measure metrics for evaluating the prediction performance of classifiers based on machine learning. To the context of abnormal behavior detection, we use four standard metrics: accuracy, true positive rate, false positive rate, and F-measure. These metrics are defined based on the *true positive* (TP), *true negative* (TN), *false positive* (FP), and *false negative* (FN).

In a malware detection task, TP denotes the count of correctly identified malicious apps, TN denotes the count of correctly identified benign apps, FP denotes the count of misclassified benign apps, and FN denotes the count of misclassified malicious apps. True positive rate (TPR) measures the proportion of the correctly identified malicious apps to all malicious apps,

$$\text{TPR} = \frac{\text{TP}}{(\text{TP} + \text{FN})}. \tag{5}$$

False positive rate (FPR) measures the proportion of the misidentified benign apps to all benign apps,

$$\text{FPR} = \frac{\text{FP}}{(\text{FP} + \text{TN})}. \tag{6}$$

Accuracy (ACC) measures the proportion of the correctly identified malware and benign apps to all apps,

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{(\text{TP} + \text{TN} + \text{FP} + \text{FN})}. \tag{7}$$

F-measure is a compromised measure that combine precision and recall and the harmonic mean of precision and recall,

$$F\text{-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{(\text{Precision} + \text{Recall})}, \tag{8}$$

Input: F: all sensitive data flows; V_b: data-flow vectors of benign apps; V_m: data-flow vectors of malicious apps; t: the threshold-difference
Output: CF, a set of critical sensitive data flows
(1) **Let** $countInSet(f, Y)$ return the total count of elements which contain sensitive data flow f in data-flow vector set Y
(2) **Let** $card(Y)$ return the count of elements in data-flow vector set Y
(3) **procedure** CFlowSel(F, V_b, V_m, t)
(4) CF $\leftarrow \varnothing$
(5) **for** *each data flow* f *in* F **do**
(6) $N_b \leftarrow countInSet(f, V_b)$
(7) $N_m \leftarrow countInSet(f, V_m)$
(8) $b_f \leftarrow N_b/card(N_b)$
(9) $m_f \leftarrow N_m/card(N_m)$
(10) **if** $|m_f - b_f| > t$ **then**
(11) CF $\leftarrow$ CF $\cup \{f\}$
(12) **end if**
(13) **end for**
(14) **return** CF
(15) **end procedure**

Algorithm 1: CFlowSel.

where in this definition Precision is the proportion of correctly identified malicious apps to all classified malicious apps,

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}, \tag{9}$$

and Recall is referred to as true positive rate.

In the above metrics, TPR refers to the ability of a classifier to correctly detect malware, FPR refers to the ability of a classifier to misclassify benign apps as malware and ACC refers to the ability of a classifier to correctly identify both benign apps and malware. F-measure, a weighted average of precision and recall, is usually used to evaluate the performance of binary classification problem on unbalanced dataset, like our abnormal behavior detection problem involving a relatively large amount of malware comparing to that of benign apps. Higher values of true positive rate and accuracy and lower value of false positive rate indicate higher detection performance. An F-measure close to 1 indicates good performance on correctly classifying the malicious apps.

We also use the area under the ROC curve (AUC) to evaluate the classification performance. AUC is a strong predictor of the overview performance of a classifier, especially for imbalanced data classification problems. This value is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. An area of 1.0 indicates a perfect classifier. Meanwhile, an area of 0.5 indicates a poor classifier.

4.3. Datasets. For all experiments, we consider a dataset of real Android applications and real malware, which is composed of the MUDFLOW dataset and the DREBIN dataset. The MUDFLOW dataset consists of 2800 benign apps and 15097 malicious apps. In these apps, benign apps contain the most popular free applications in Google Play Store and malicious apps contain malware published by Genome project [18] and VirusShare [19] database. The DREBIN dataset contains 5560 malicious apps.

All sensitive data flows within MUDFLOW dataset are extracted by FLOWDROID with proper settings. For 5560 malicious apps contained in DREBIN dataset, 1083 of which them not analyzable (375 result in exceptions and 708 exceed time/memory limit). In the remaining 4477 malicious apps, 302 apps contain no sensitive data flows according to the FLOWDROID's analysis results. Therefore, the final DREBIN dataset used for experiments actually contains 4175 malicious apps.

In the following, let MW denote the MUDFLOW dataset. DN denotes a new dataset which consists of benign apps from MUDFLOW and the analyzable malware from DREBIN.

5. Evaluations and Discussions

To evaluate our approach, we conduct experiments to answer the following research questions with the experimental results.

RQ1. Can SCDFLOW effectively reduce the dimension of sensitive data-flow features?

To evaluate the effectiveness of *CFlowSel*, we summarize the count of critical data-flow features in each category, respectively, on dataset MW and dataset DN and the results are shown in Tables 1 and 2. In our experiments, the count of all sensitive data flows is 3733. From Tables 1 and 2, we find that our algorithm is able to remove irrelevant sensitive data flows and greatly reduce the count of data-flow features in each source category. The choices on the value of threshold-difference t ($t = 0.08$ for dataset MW and $t = 0.11$ for dataset DN) are discussed lately in Figure 4.

Table 1: The count of critical sensitive data flows in each category on dataset *MW* when $t = 0.08$.

Category	The count of data-flow features
LOCATION_INFORMATION	82
NETWORK_INFORMATION	64
DATABASE_INFORMATION	76
UNIQUE_IDENTIFIER	66
CONTENT_RESOLVER	91
NO_SENSITIVE_SOURCE	55
CALENDAR_INFORMATION	86
BLUETOOTH_INFORMATION	104
ACCOUNT_INFORMATION	88
FILE_INFORMATION	126

Table 2: The count of critical sensitive data flows in each category on dataset *DN* when $t = 0.11$.

Category	The count of data-flow features
LOCATION_INFORMATION	79
NETWORK_INFORMATION	66
DATABASE_INFORMATION	87
UNIQUE_IDENTIFIER	77
NO_SENSITIVE_SOURCE	55
CALENDAR_INFORMATION	65
BLUETOOTH_INFORMATION	95
ACCOUNT_INFORMATION	113

SCDFLOW can effectively reduce the dimension of sensitive data flows features.

RQ2. Can SCDFLOW improve the detection performance of per-category outlier detection?

As described in Section 3.3, the overall SCDFLOW classifier depends on individual calculated outlier scores of outlier detection for each source category. To evaluate the influence of considering critical data flows as feature to the accuracy of these detectors, we compute the AUC value for each outlier detector. The performance comparison between SCDFLOW and MUDFLOW of precategory outlier detection is shown in Tables 3 and 4. N_b denotes the count of benign apps and N_m denotes the count of malicious apps. AUC of all flows denotes the AUC value of outlier detector that takes all sensitive data flows as feature. AUC of critical flows denotes the AUC value of outlier detector that takes critical sensitive data flows as feature. We remove categories in which the count of benign apps and malware is small. Critical data-flow features from these categories may be inaccurate because of too few application instances.

From Tables 3 and 4, we observe that, on both dataset MW and dataset DN, taking these critical data flows as feature can improve the performance of outlier detectors on most categories. Some data flows are irrelevant and noisy data-flow features which frequently emerge in both benign apps and malware. Removing these data-flow features can increase the distance between malware and benign apps, which makes the outlier score of each malware increase. In Table 4, SCDFLOW records some reduction on AUC values. We further analyze apps in these categories and find that some malware and benign apps contain no data flow from critical sensitive data flows, which makes these apps undetectable and degrades the performance of these outlier detectors.

SCDFLOW is able to improve the performance of per-category outlier detection.

RQ3. Can SCDFLOW effectively improve the malware detection rate?

To evaluate the classification performance of SCDFLOW, we compare SCDFLOW with MUDFLOW on datasets MW and DN in terms of measure metrics. On each dataset, we adopt 10-fold cross validation to access the results of classifier and repeat the following experiment ten times:

(i) We trained the classifier on the feature vectors from a random 90% of the benign apps.

(ii) The remaining 10% of benign apps as well as the whole malicious apps form test set.

On the two datasets, cross validation tests for 2-, 3-, and 5-folds are also performed. The average results with ν-SVM configured as $\nu = 0.15$ on different measure metrics are, respectively, shown in Tables 5 and 6.

From Tables 5 and 6, we observe that, on both datasets MW and DN, SCDFLOW improves the ACC and TPR of MUDFLOW and causes little effect on FPR. The AUC value of SCDFLOW is higher than MUDFLOW on both datasets, which illustrates the overall classification precision of SCDFLOW outperforms MUDFLOW. The F-measure value of SCDFLOW is higher than MUDFLOW on both datasets, which illustrates that SCDFLOW improves the malware detection precision of MUDFLOW. Let p denote the value of folds of cross validation tests. For different p, Tables 5 and 6 show that the AUC value remains almost unchanging and the F-measure value increases with the variation of p. This variation rule reflects that the overview classification performance subjects to little influence by the amount of training data and the performance of malware classification increases with the increase of training data.

SCDFLOW effectively improves the malware detection rate by 5.73%~9.07% on the two datasets.

RQ4. How about the memory consumption of SCDFLOW when compared with MUDFLOW?

SCDFLOW takes *CFlowSel* algorithm to generate critical sensitive data flows and greatly improves the operational efficiency of MUDFLOW. This *CFlowSel* algorithm is implemented in R scripts. SCDFLOW adopts four steps to perform abnormal behavior detection. In the first step, *App processor* uses FLOWDROID to extract information flows whose memory consumption is beyond the scope of this

Table 3: Performance comparison of precategory outlier detection on dataset *MW*.

Category	N_b	N_m	AUC of all flows	AUC of critical flows
LOCATION_INFORMATION	1168	6410	0.9028	0.9338
NETWORK_INFORMATION	2254	14086	0.9414	0.9598
DATABASE_INFORMATION	1525	7648	0.9508	0.9689
UNIQUE_IDENTIFIER	972	11456	0.9360	0.9751
CONTENT_RESOLVER	1067	3771	0.9494	0.9632
NO_SENSITIVE_SOURCE	2790	14261	0.9323	0.9557
CALENDAR_INFORMATION	1275	6928	0.9323	0.9557
BLUETOOTH_INFORMATION	183	283	0.9643	0.9651
ACCOUNT_INFORMATION	320	542	0.5138	0.5340
FILE_INFORMATION	563	1358	0.9338	0.9506

Table 4: Performance comparison of precategory outlier detection on dataset *DN*.

Category	N_b	N_m	AUC of all flows	AUC of critical flows
LOCATION_INFORMATION	1168	551	0.9915	0.9985
NETWORK_INFORMATION	2254	3068	0.9953	0.9954
DATABASE_INFORMATION	1525	815	0.9978	0.9989
UNIQUE_IDENTIFIER	972	1888	0.9942	0.9984
NO_SENSITIVE_SOURCE	2790	3883	0.8973	0.9177
CALENDAR_INFORMATION	1275	499	0.9399	0.9892
BLUETOOTH_INFORMATION	183	139	0.9953	1.0000
ACCOUNT_INFORMATION	320	33	0.9981	0.9988

paper. We perform MUDFLOW and SCDFLOW on datasets MW and DN for 10 times and count average memory consumption of *Outlier Detection*, *Feature Vector Generator,* and *Classification*. The memory consumption of each step is compared in Figure 3. From Figure 3, we find, on both datasets, SCDFLOW causes ignorable memory consumption increase on *Outlier Detection* step and little influence on *Feature Vector Generator* and *Classification*.

> Compared with MUDFLOW, SCDFLOW causes ignorable increase on memory consumption.

RQ5. Is *CFlowSel* more effective in selecting critical data-flow features when compared with mutual information and chi-square?

To evaluate the effectiveness of *CFlowSel*, we firstly compare three feature selection algorithms with the same count of data-flow features on dataset MW. We select features with *CFlowSel* when $t = 0.02,\ 0.04,\ 0.06,\ 0.08$; thus we automatically have four sets of critical data-flow sets. The counts of data-flow features on each source category are different. For comparison, three feature selection algorithms must select the same count of data-flow features on every source category. We sort all data flows with mutual information and chi-square and manually select top data flows in ranked lists. Finally, we build the abnormal detection models based on these critical data flows and the results are shown in Table 7. In Table 7, $\text{SCDFLOW}_{\text{mutual information}}$, $\text{SCDFLOW}_{\text{chi-square}}$, and $\text{SCDFLOW}_{CFlowSel}$, respectively, denote SCDFLOW adopting feature selection algorithm mutual information, chi-square, and *CFlowSel*. Different values of t in $\text{SCDFLOW}_{\text{mutual information}}$ and $\text{SCDFLOW}_{\text{chi-square}}$ mean that mutual information and chi-square select the same counts of data-flow features as *CFlowSel* selects the specific t.

From Table 7, we observe that, with the increase of threshold t, the count of data flows decreases, AUC and F-measure of mutual information and chi-square decrease, and measure metrics of *CFlowSel* increase. This variation rule indicates that *CFlowSel* is more effective than mutual information and chi-square for abnormal behavior detection. When $t = 0.02$, the measure metrics of mutual information show similar classification performance when compared with *CFlowSel*. With the reduction of the count of data flows, mutual information and chi-square select less critical data flows and degrade the classification performance; meanwhile, *CFlowSel* still selects more critical data flows and improves the detection performance. Compared with the other two algorithms, *CFlowSel* achieves higher AUC and F-measure with the same count of critical data-flow features.

The threshold t in *CFlowSel* influences the malware detection performance of abnormal behavior detection. In this experiment, we take F-measure to evaluate the classification performance of malware detection. For illustrating the influence of threshold t with measure metric F-measure, we conduct a series of experiments on dataset MW and dataset DN. We, respectively, set t with a set of values and compute F-measure for each value of t. We plot the continuous distribution of F-measure with the values of t in Figure 4.

Figure 4 shows the variation rule of F-measure with t. With the increase of t, *CFlowSel* selects more critical data-flow features which increases F-measure. As t continues to

Table 5: Measure metrics comparison on dataset *MW* when $t = 0.08$.

Tool	Folds	ACC	TPR	FPR	AUC	*F*-measure
MUDFLOW	2	85.48%	85.73%	17.09%	0.9116	0.9153
	3	85.70%	85.99%	19.06%	0.9115	0.9187
	5	85.55%	85.68%	17.75%	0.9116	0.9196
	10	85.57%	85.76%	17.94%	0.9113	0.9225
SCDFLOW	2	90.48%	91.33%	18.62%	0.9503	0.9476
	3	91.00%	91.70%	20.46%	0.9504	0.9501
	5	91.12%	91.53%	19.87%	0.9504	0.9521
	10	91.29%	91.49%	18.21%	0.9503	0.9545

Table 6: Measure metrics comparison on dataset *DN* when $t = 0.11$.

Tool	Folds	ACC	TPR	FPR	AUC	*F*-measure
MUDFLOW	2	84.48%	83.29%	16.71%	0.9178	0.8988
	3	84.43%	83.40%	17.08%	0.9208	0.9008
	5	84.52%	84.86%	17.98%	0.9208	0.9062
	10	84.71%	84.90%	18.09%	0.9176	0.9125
SCDFLOW	2	91.32%	95.46%	21.05%	0.9752	0.9427
	3	91.83%	94.67%	20.06%	0.9750	0.9507
	5	92.44%	93.93%	18.66%	0.9749	0.9563
	10	93.18%	93.97%	18.60%	0.9749	0.9627

increase, *CFlowSel* selects too less data-flow features which decreases *F*-measure. With t continuing to increase, SCDFLOW performs worse than the original MUDFLOW. This is because, with insufficient data-flow features, the distinguishability between benign apps and malware is lost. On dataset MW and dataset DN, when t, respectively, equals 0.08 and 0.11, *F*-measure reaches the maximum 0.9545 and 0.9627. The value of t is not an optimal value for all datasets, since each dataset has its own optimal t. From Figure 4, we also find that the optimal value of t in dataset MW can be used to select features which are able to improve the classification performance on dataset DN, and vice versa.

Malware detection performance is also influenced by the count of data-flow features of mutual information and chi-square. For illustrating the influence of the count of data-flow features selected by mutual information and chi-square, we conduct a series of experiments on the two datasets. We, respectively, set the count of data-flow features with a set of values and evaluate *F*-measure for each count of data-flow features. We plot the continuous distribution of *F*-measure in Figure 5.

Figure 5 shows the variation rule of *F*-measure with the count of data-flow features. For mutual information, when the count of data-flow features equals 200, *F*-measure reaches the maximum 0.9419 on dataset MW and 0.9521 on dataset DN. For chi-square, when the count of data-flow features equals 400 on dataset MW and 100 on dataset DN, *F*-measure, respectively, reaches the maximum 0.9387 on dataset MW and 0.9489 on dataset DN.

From Figures 4 and 5, we observe that, on both datasets, the maximum *F*-measure value of *CFlowSel* outperforms the values of mutual information and chi-square. Combined with Tables 1 and 2, *CFlowSel* selects less data-flow features which are more effective at abnormal behavior detection.

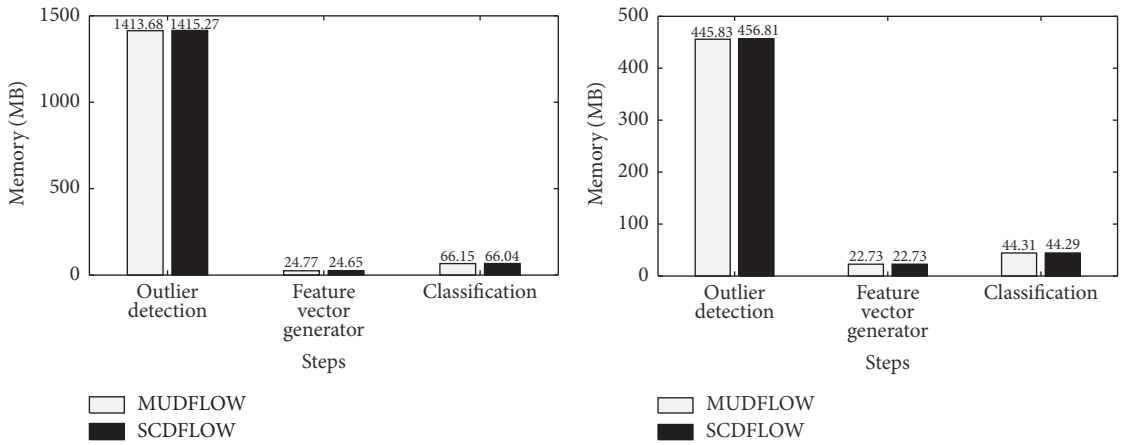
> *CFlowSel* is more effective when compared with mutual information and chi-square in terms of classification performance on the same count of data-flow features, the maximum *F*-measure value, and the selection of critical data flows.

RQ6. Can benign apps of 2014 be still representative of nowadays "normal" usage of sensitive data?

To illustrate the representativeness of benign apps in 2014, we use recently benign apps from Google Play to verify the effectiveness of SCDFLOW. For each of the 28 app categories in Google Play store, we download the top 110 most popular free applications as of May 2016. Despite the suffering from several download failures, this gives us a total of 3000 apps. In our experimental environment, we only obtain 328 analyzable benign apps with sensitive data flows. For the 328 benign apps, 269 are classified as benign and 59 are misclassified as malware. The proportion of correctly identified benign apps is 82.01%. This indicates that most benign apps own the recently "normal" usage of sensitive data. We further analyze those misclassified benign apps, which are mainly caused by performing abnormal usage of sensitive data. For a misclassified app, Android IROAD, Table 8 shows its usage of sensitive data from NETWORK_INFORMATION category. Flows to WifiManager.setWifiEnabled(), WifiManager.enableNetwork(), and Activity.startActivity() lead to the

Table 7: Measure metrics comparison on dataset *MN* with different threshold.

Tool	Comparison					
	t	ACC	TPR	FPR	AUC	F-measure
SCDFLOW$_{mutual\ information}$	0.02	89.13%	89.28%	19.09%	0.9348	0.9416
	0.04	88.12%	88.26%	19.37%	0.9313	0.9359
	0.06	85.84%	85.95%	20.09%	0.9153	0.9226
	0.08	83.18%	83.17%	16.49%	0.8970	0.9066
SCDFLOW$_{chi\text{-}square}$	0.02	87.68%	87.81%	19.61%	0.9297	0.9333
	0.04	83.82%	83.86%	18.37%	0.9035	0.9105
	0.06	78.04%	77.92%	15.43%	0.8817	0.8745
	0.08	74.28%	74.10%	15.69%	0.8575	0.8488
SCDFLOW$_{CFlowSel}$	0.02	88.96%	89.09%	18.26%	0.9351	0.9406
	0.04	90.83%	91.01%	18.65%	0.9498	0.9512
	0.06	90.94%	91.12%	18.79%	0.9491	0.9518
	0.08	91.32%	91.53%	19.93%	0.9504	0.9539



(a) Memory consumption on dataset MW

(b) Memory consumption on dataset DN

Figure 3: Memory consumption comparison.

abnormal behaviors. As stated in descriptions, this app needs to connect Wifi to play video recording.

> Benign apps we use are representative of nowadays "normal" usage of sensitive data.

6. Malware Case Study

To evaluate the effectiveness of SCDFLOW, we consider three well-known malware families, namely, DroidKungFu [20], GoldDream [21], and GingerMaster [22]. For each sample of these families, we present the processing of SCDFLOW and illustrate how this tool works to detect those malware.

DroidKungFu tries to exploit several vulnerabilities to gain root access and steal sensitive data from the device. This malware would collect a variety of information on the infected mobile phone, including the IMEI number, phone model, and the Android OS version, and send these sensitive data to a hard-coded remote server. First, *App Preprocessor* uses FLOWDROID to extract all sensitive data flows within this malware sample. Table 9 shows some flows in *DroidKungFu* malware sample at SuSi category level. Based on these flows, this sample is classified into source categories DATABASE_INFORMATION, UNIQUE_IDENTIFIER, NETWORK_INFORMATION, and CALENDAR_INFORMATION to be processed by *Outlier Detection in Source Category*. Outlier scores of this sample are, respectively, 0.728, 0.705, 0.706, and 0.704. The value of outlier score indicates the maliciousness of this malware in that category. *Feature Vector Generator* aggregates outlier scores across all source categories and generates this malware's feature vector {0, 0, 0.704, 0, 0, 0.728, 0, 0, 0.706, 0, 0, 0, 0.705}. Finally, based on its feature vector, *Classification* uses the trained one-class classification model to identify this sample as "malicious." The predicted value of v-SVM one-class classifier is −33.332, which tells how confident the decision is. The larger its absolute value is, the higher the confidence of this sample is malware.

GoldDream is a Trojan which registers a receiver so that it will be notified for system events such as SMS_RECEIVED or when there is an incoming/outgoing phone call. Upon these

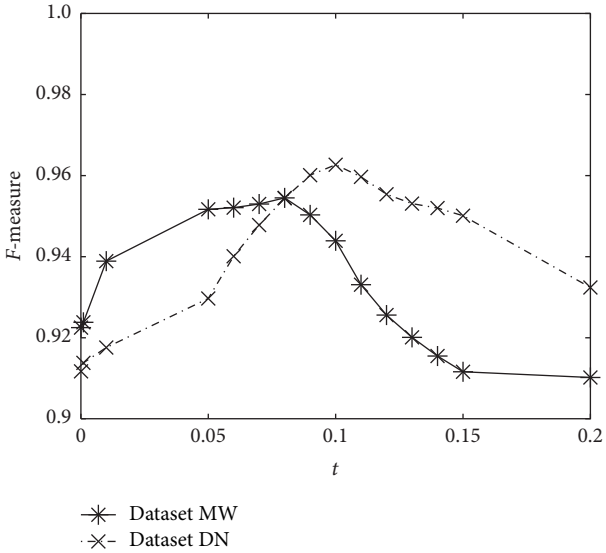


Figure 4: F-measure of SCDFLOW when varying the value t on the two datasets.

Table 8: Flows in Android *IROAD* app.

WifiManager.getConfiguredNetworks() ⇝ WifiManager.setWifiEnabled()
WifiManager.getConfiguredNetworks() ⇝ WifiManager.enableNetwork()
WifiManager.getConfiguredNetworks() ⇝ Log.d()
WifiManager.getScanResults() ⇝ WifiManager.setWifiEnabled()
WifiManager.getScanResults() ⇝ WifiManager.enableNetwork()
WifiManager.getScanResults() ⇝ Activity.startActivity()
WifiManager.getScanResults() ⇝ Log.d()
WifiManager.getConnectionInfo() ⇝ Log.d()
WifiManager.getSSID() ⇝ Log.d()
WifiManager.getNetworkInfo() ⇝ Log.d()
ConnectivityManager.getNetworkInfor() ⇝ Log.d()
ConnectivityManager.getActiveNetworkInfo() ⇝ Log.d()

events, the malware launches a background service without user's knowledge. Once the service gets started, this malware will collect a variety of information on the infected mobile phone, including the IMEI number as well as the unique subscriber ID and then upload this information to a remote server. This sample is processed by similar processing steps. Table 10 shows some flows in *GoldDream* malware sample by SuSi category level. Outlier scores on source categories DATABASE_INFORMATION, UNIQUE_IDENTIFIER, and NETWORK_INFORMATION are, respectively, 0.718, 0.732, and 0.703. The final predicted value is −49.444, which

Table 9: Flows in *DroidKungFu* malware sample.

DATABASE_INFORMATION ⇝ INTENT
DATABASE_INFORMATION ⇝ LOG
DATABASE_INFORMATION ⇝ FILE
DATABASE_INFORMATION ⇝ NETWORK
UNIQUE_IDENTIFIER ⇝ INTENT
UNIQUE_IDENTIFIER ⇝ LOG
UNIQUE_IDENTIFIER ⇝ FILE
UNIQUE_IDENTIFIER ⇝ NETWORK
NETWORK_INFORMATION ⇝INTENT
NETWORK_INFORMATION ⇝ LOG
NETWORK_INFORMATION ⇝ FILE
NETWORK_INFORMATION ⇝ NETWORK
CALENDAR_INFORMATION ⇝ LOG

Table 10: Flows in *GoldDream* malware sample.

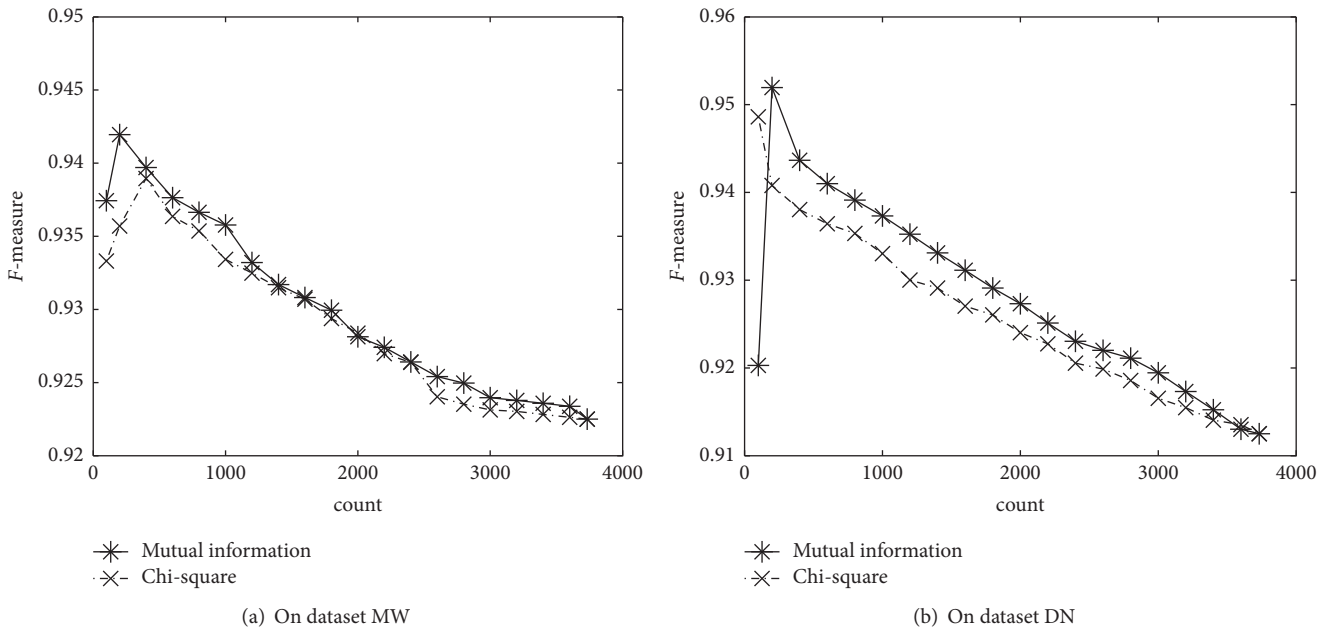
DATABASE_INFORMATION ⇝ CONTENT_RESOLVER
DATABASE_INFORMATION ⇝ NETWORK
DATABASE_INFORMATION ⇝ LOG
DATABASE_INFORMATION ⇝ FILE
DATABASE_INFORMATION ⇝ INTENT
UNIQUE_IDENTIFIER ⇝ NETWORK
NETWORK_INFORMATION ⇝ SMS_MMS
NETWORK_INFORMATION ⇝ FILE

Table 11: Flows in *GingerMaster* malware sample.

DATABASE_INFORMATION ⇝ LOG
DATABASE_INFORMATION ⇝ NETWORK
DATABASE_INFORMATION ⇝ FILE
DATABASE_INFORMATION ⇝ INTENT
UNIQUE_IDENTIFIER ⇝ NETWORK
UNIQUE_IDENTIFIER ⇝ INTENT
NETWORK_INFORMATION ⇝ NETWORK
NETWORK_INFORMATION ⇝ INTENT
NETWORK_INFORMATION ⇝ LOG
LOCATION_INFORMATION ⇝ LOG

indicates that this sample is detected as malware with high confidence.

GingerMaster is also a Trojan application that is repackaged into legitimate apps. This malware would silently launch a service in the background. This background service will accordingly collect various information including the device ID, phone number, and others (e.g., by reading/proc/cpuinfo) and then upload them to remote server. SCDFLOW processes this sample with similar procedures. Table 11 shows the flows in *GingerMaster* malware sample by SuSi category level. Outlier scores on source categories DATABASE_INFORMATION, UNIQUE_IDENTIFIER, NETWORK_INFORMATION, and LOCATION_INFORMATION are, respectively, 0.831, 0.827, 0.829, and 0.849. These scores



(a) On dataset MW

(b) On dataset DN

Figure 5: F-measure of SCDFLOW when varying the count of data-flow features.

indicate this malware is more malicious in these categories. The final predicted value of this sample is −53.862.

7. Threats to Validity

In this section, the main threats to validity in the process of our study are proposed.

Since SCDFLOW mainly focuses on improving the classification performance of MUDFLOW, it is also subject to the same limitations and threats as described in MUDFLOW. Meanwhile, SCDFLOW takes the same analysis setting of FLOWDROID as MUDFLOW chooses. This kind of analysis setting trades a small amount of precision in favor of a significant speed gain, extracts imprecise data flows, and therefore leads to selecting inaccurate critical sensitive data flows. As deficiencies of static analysis, FLOWDROID encounters an invalidation when performing taint analysis on malware that combines reflection, native code, self-decrypting code, or other obfuscation techniques with sensitive data usage.

In this paper, SCDFLOW uses sensitive data flows to describe the sensitive data usage behavior of Android applications and aims to identify malware based on its abnormal data flows. Therefore, SCDFLOW becomes invalid when encountering malware leaking no sensitive data based on FLOWDROID analysis results. We do not know currently popular Android malware, its code features, and its obfuscation features. Our experiment results should be seen as a result on a publicly available benchmark. False negative and false positive rates of SCDFLOW may be potentially high on some other datasets.

SCDFLOW, which trains a one-class classifier on benign applications, aims to report potential novel malware instances. The effectiveness of this method depends on the representative of "benign" apps. The benign apps used in MUDFLOW datasets consist of top 100 most popular free applications from 30 app categories of March 1, 2014. We conduct an experiment to illustrate that the classifier training from benign apps of 2014 is still representative of nowadays "normal" usage of benign apps. The number of recently analyzable benign apps may influence the effectiveness of this experiment.

8. Related Work

In this paper, we study the difference on occurrence frequency of data flows between malware and benign apps, select critical data flows based on these differences, and take these flows as features to improve the performance of abnormal behavior detection. This work is related to approaches that leverage information flow analysis to identify sensitive data leakages, the ones that detect Android malware based on machine learning, as well as the ones that select most informative features to implement highly precise malware detection.

8.1. Information Flow Analysis for Android Applications. As mobile devices are a ubiquitous source of private and confidential data, it is not a surprise that many security research work has developed taint analysis techniques to identify sensitive data leakages. FLOWDROID is a sophisticated analysis tool that leverages static taint analysis to detect sensitive information leakages among Android apps. This tool provides a high precise context-, flow-, field-, and object-sensitive and lifecycle-aware taint analysis, which tracks the data flow of taint data from sensitive source to sink. It reports all sensitive data flows from predefined source API methods to sink API methods within Android apps. Prior work like AndroidLeaks [23] also takes static taint analysis to perform

intracomponent information leakages detection but provides imprecise analysis results. Intercomponent sensitive information leakages have been researched by several methods [24, 25]. IccTA [25] models intercomponent communication and seeks to identify sensitive intercomponent and interapplication information flows.

Other techniques focus on leveraging dynamic taint analysis techniques to identify sensitive information leakages. TAINTDROID [26] modifies the Android Dalvik virtual machine to monitor the transmission of sensitive information within Android apps at runtime. Our approach is orthogonal to all these analysis techniques and only considers sensitive data flows extracted by FLOWDROID to characterize the information leakage behavior of Android applications.

8.2. Android Malware Detection. As for malware posing a threat to the security of the Android platform, malware detection has received many attentions. Machine learning techniques have been used by several approaches [27–30] in Android malware detection. DroidAPIMiner [27] extracts fine-grained API method information from applications and adopts supervised learning algorithm to detect malware. DroidMiner [29] automatically extracts robust and fine-grained application behavior into a sequence of threat modalities and implements effective and precise malware detection and malware family classification through machine learning. AppContext [30] identifies malware based on the context of security-sensitive API call. It constructs call graph and generates contexts by leveraging predefined activation events and context factors for each Android application. Then, it takes the extracted contexts as features to detect malicious behaviors. All these methods train classifier on malware samples and can therefore be effective at detecting other samples of similar malware. Differently, SCDFLOW trains classifier on benign apps, aiming at detecting novel malware based on its abnormal data usage behavior. However, these techniques that perform static analyses to implement precisely known malware detection are complementary to SCDFLOW. As a malware detection method, DroidSIFT [31] is able to detect novel malware. DroidSIFT only focuses on permission-related critical API calls. Meanwhile, SCDFLOW focuses on sensitive data flows and describes the behavior of applications by these flows.

Several malware detection approaches take feature selection methods to select most informative features [14, 32, 33]. These approaches often employ methods, for example, mutual information, Pearson correlation coefficient, and T-test to measure the relevance of features and class variable. Wang et al. [14] explore most informative permissions and implement high precision malware detection based on these permissions. However, SCDFLOW selects critical data-flow features based on their difference on occurrence frequency between malware and benign apps.

9. Conclusion

In this paper, we find that some sensitive data flows frequently emerge among benign apps and malware. These data flows contribute less for malware detection based on abnormal data flows. We present an approach, SCDFLOW, which can select critical data flows and take these flows as features to detect malware based on abnormal data flows. The novel algorithm *CFlowSel* proposed by SCDFLOW can select critical data flows based on their occurrence frequencies between benign apps and malware. Through experiments, we verify that *CFlowSel* outperforms two existing feature selection algorithms. We also show that this algorithm can effectively reduce the count of data-flow features for abnormal detection on the two datasets. Removing irrelevant and noisy data flows and considering critical data flows as features will improve the precision of malware identification based on abnormal data flows. SCDFLOW, compared with MUDFLOW, effectively improves the malware detection rate by 5.73% on dataset *MW* and 9.07% on dataset *DN* and causes ignorable increase on memory consumption. Although our approach mainly focuses on the malicious behavior of sensitive information leakages, *CFlowSel* algorithm can actually be used to other features, like requested permissions, API calls, and so forth.

Despite the effectiveness of SCDFLOW, there are still lots of opportunities for improvement. Our future work will focus on addressing the following problems. SCDFLOW does not consider information leakages across multiple components within Android applications, which makes the description of information leakage behavior inaccurate. We will combine taint analysis tool that can precisely model intercomponent communications (ICC) to extract sensitive data flows across multiple components and multiple apps. To overcome the limitations of static taint analysis, we are investigating to use a hybrid approach that combines static and dynamic taint analysis. This will provide a better view for the usage of sensitive data in Android applications and will consequently increase the precision of malware detection based on abnormal data flow.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National High Technology Research and Development Program (863 Program) of China (no. 2015AA017203), the National Natural Science Foundation of China (nos. 61303033, 61502368, and 61602537), the Key Program of NSFC (no. U1405255), the Natural Science Basis Research Plan in Shaanxi Province of China (no. 2016JM6034), China 111 Project (no. B16037), and the Special Research Foundation of MIIT (no. MJ-2014-S-37).

References

[1] Smartphone os market share, 2016 q2, http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[2] http://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf.

[3] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: detecting malicious apps in official and alternative

android markets," in *Proceedings of the NDSS Symposium*, get off of my market, 2012.

[4] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: automatic security analysis of smartphone applications," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY '13)*, pp. 209–220, ACM, February 2013.

[5] A. Reina, A. Fattori, and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *Proceedings of the European Workshop on System Security (EuroSec '13)*, April 2013.

[6] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 235–245, ACM, November 2009.

[7] 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 (CCS '11)*, pp. 627–638, ACM, October 2011.

[8] 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)*, pp. 281–294, ACM, June 2012.

[9] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: effective and explainable detection of android malware in your pocket," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS '14)*, San Diego, Calif, USA, February 2014.

[10] W.-C. Wu and S.-H. Hung, "DroidDolphin: a dynamic android malware detection framework using big data and machine learning," in *Proceedings of the Conference on Research in Adaptive and Convergent Systems (RACS '14)*, pp. 247–252, ACM, October 2014.

[11] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-sandbox: combining static and dynamic analysis with machine-learning techniques," *International Journal of Information Security*, vol. 14, no. 2, pp. 141–153, 2015.

[12] V. Avdiienko, K. Kuznetsov, A. Gorla et al., "Mining apps for abnormal usage of sensitive data," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE '15)*, pp. 426–436, IEEE, Florence, Italy, May 2015.

[13] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pp. 29–38, ACM, August 2003.

[14] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014.

[15] S. Arzt, S. Rasthofer, C. Fritz et al., "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*, pp. 259–269, ACM, June 2014.

[16] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing android sources and sinks," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '14)*, February 2014.

[17] P.-H. Chen, C.-J. Lin, and B. Schölkopf, "A tutorial on ν-support vector machines," *Applied Stochastic Models in Business and Industry*, vol. 21, no. 2, pp. 111–136, 2005.

[18] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '12)*, pp. 95–109, San Francisco, Calif, USA, May 2012.

[19] http://virusshare.com.

[20] https://www.csc2.ncsu.edu/faculty/xjiang4/DroidKungFu.html.

[21] https://www.csc2.ncsu.edu/faculty/xjiang4/GoldDream/.

[22] https://www.csc2.ncsu.edu/faculty/xjiang4/GingerMaster/.

[23] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *Proceedings of the 5th International Conference on Trust and Trustworthy Computing (TRUST '12)*, pp. 291–307, Vienna, Austria, June 2012.

[24] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, pp. 1–6, ACM, June 2014.

[25] L. Li, A. Bartel, T. F. D. A. Bissyande et al., "IccTA: detecting inter-component privacy leaks in android apps," in *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE '15)*, Florence, Italy, May 2015.

[26] W. Enck, P. Gilbert, S. Han et al., "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the in 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, pp. 393–407, Vancouver, Canada, 2010.

[27] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: mining API-level features for robust malware detection in Android," in *Security and Privacy in Communication Networks*, pp. 86–103, Springer, 2013.

[28] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," in *Proceedings of the 39th IEEE Annual Computer Software and Applications Conference (COMPSAC '15)*, vol. 2, pp. 179–184, IEEE, July 2015.

[29] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: automated mining and characterization of fine-grained malicious behaviors in android applications," in *Computer Security—ESORICS 2014*, pp. 163–182, Springer, 2014.

[30] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE '15)*, vol. 1, pp. 303–313, IEEE, May 2015.

[31] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS '14)*, pp. 1105–1116, Scottsdale, Ariz, USA, November 2014.

[32] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.

[33] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2015.