2012 Seventh Asia Joint Conference on Information Security

DroidMat: Android Malware Detection through Manifest and API Calls Tracing

Dong-Jie Wu[1], Ching-Hao Mao[2], Te-En Wei[1], Hahn-Ming Lee[1,3], Kuo-Ping Wu[1]
[1]Dept. of Computer Science and Information Engineering,
National Taiwan University of Science and Technology, Taipei, Taiwan
[2]Institute for Information Industry, Taipei, Taiwan
[3]Institute of Information Science, Academia Sinica, Taipei, Taiwan
Email: M9915013@mail.ntust.edu.tw

***Abstract*—Recently, the threat of Android malware is spreading rapidly, especially those repackaged Android malware. Although understanding Android malware using dynamic analysis can provide a comprehensive view, it is still subjected to high cost in environment deployment and manual efforts in investigation.**

In this study, we propose a static feature-based mechanism to provide a static analyst paradigm for detecting the Android malware. The mechanism considers the static information including permissions, deployment of components, Intent messages passing and API calls for characterizing the Android applications behavior. In order to recognize different intentions of Android malware, different kinds of clustering algorithms can be applied to enhance the malware modeling capability.

Besides, we leverage the proposed mechanism and develop a system, called *DroidMat*. First, the *DroidMat* extracts the information (e.g., requested permissions, Intent messages passing, etc) from each application's manifest file, and regards components (Activity, Service, Receiver) as entry points drilling down for tracing *API Calls* related to permissions. Next, it applies K-means algorithm that enhances the malware modeling capability. The number of clusters are decided by *Singular Value Decomposition (SVD)* method on the low rank approximation. Finally, it uses kNN algorithm to classify the application as benign or malicious.

The experiment result shows that the recall rate of our approach is better than one of well-known tool, Androguard, published in Blackhat 2011, which focuses on Android malware analysis. In addition, *DroidMat* is efficient since it takes only half of time than Androguard to predict 1738 applications as benign or malicious.

***Index Terms*—Smartphone security, Android malware, static analysis, feature-based, anomaly detection**

I. Introduction

With the development of scientific and technological progress, today's mobile phones have evolved into *"smartphones"* with more sophisticated functionalities than before. A recent report from Gartner [5] shows that there are 149 millions of smartphones sold in the fourth quarter of 2011 with 47 percent growth. Also, the number of smartphone applications is explosively growing. Another report from Lookout [6] mentions that the Android market surpassed 400000 app mark in December 2011 doubling the market total in just 8 months. Not only official marketplaces (Google Play) but a number of third-party ones (e.g. AppBrain) provide smartphone applications, which boost the popularity. Unfortunately, such popularity also attracts the malware developers. The smartphone becomes another hotbed for malware doing malicious activities.

The Android market is the fastest growing mobile application platform. A recent report from Lookout shows that the Android Market is growing at three times the rate of Apple's App store [7]. However, unlike Apple's App store which may check each available application manually by software security experts, there is lack of complete app inspecting process before the applications being published on the Android market. Google takes passive mechanism that allows anyone to publish applications on the Android market. If an application is reported as malware by users, it will then be removed. The openness of the Android market attracts both benign and malicious developers. Furthermore, Android allows installing third-party applications that may increase the spread of Android malware.

Android security model highly relies on permission-based mechanism [8], [18]. There are about 130 permissions that govern access to different resources. Whenever the user install a new app, he would be prompt to approve or reject all permissions requested by the application. However, users are with rare knowledge to determine if permissions might be harmful or not. They need extra information to help them making the correct choice.

In this study, we propose a feature-based mechanism to provide a static analyst paradigm for detecting the Android malware. The mechanism considers the static information including permissions, deployment of components, Intent messages passing and API calls for characterizing the Android applications behavior. In order to recognize different intentions of Android malware, different kinds of clustering algorithms can be applied to enhance the malware modeling capability.

Furthermore, we leverage the proposed mechanism and develop a system, called *DroidMat*İt extracts the information (e.g., requested permissions, Intent messages passing, etc) from each application's manifest file. In addition, it traces API calls for each component since API calls in different components may imply different intentions. Then, it applies K-means algorithm that enhances the malware modeling capability. Finally, it uses kNN algorithm to classify the application as benign or malicious.

978-0-7695-4776-3/12 $26.00 © 2012 IEEE
DOI 10.1109/AsiaJCIS.2012.18

The primary contributions of this paper are given as follows:

- providing a static feature-based mechanism to extract representative configuration and trace API calls for identifying the Android malware.
- no need for dynamic simulation that can save the cost in environment deployment and manual efforts in investigation.
- developing a system, called *DroidMat* which can help vetting applications in fully automatic way.

The rest of this paper is organized as follows: Section II provides some related works; Section III gives an overview of *DroidMat*; Section IV illustrates the evaluation results; Finally, we make our conclusion in Section V.

II. Related Work

In this section, we describe the related work about Android malware detection in different aspects. Furthermore, we also discuss different features used by other related work.

A. Misuse Detection

Misuse detection is a signature-based approach that detected malware by the sets of rules or policies. The advantage of misuse detection is that it can precisely detect the Android malware if matching any of signatures.

There are several related work applying that mechanism to detect mobile malware. Kim *et al.* [22] build a power consumption history from the collected samples, and generate a power signature from the constructed history for power-aware malware detection. Ongtang *et al.* [28] propose *Saint* to protect the interface accessible to other applications according to security policies defined by the application developers. Fuchs *et al.* [19] propose *ScanSroid* that extracts security specifications of the application, and apply data flow analysis to check if any of data flows violate them. Enck *et al.* [15] propose *Kirin* security service that perform the certification of applications. They define various of potential dangerous permission combinations as rules to block the installation of potential unsafe applications. Desnos *et al.* [11] develope algorithms to help them constructing the rules. They propose a signature-based method and also use the permission properties. Then, they construct the control flow graph through these collected contents for detecting Android malware. Zhou *et al.* [37] first propose a permission behavior to detect new Android malware and then apply heuristic filtering for detecting unknown Android malware. This hybrid method called *DroidRanger* that resolve the disadvantage of lacking ability to detect unknown malware. These are important surveyed papers that provide contributions in Android malware detection. However, misuse detection is not adaptive to the novel Android malware and always need to update the signatures.

B. Anomaly Detection

Anomaly detection is different to misuse detection, it usually applies machine learning algorithms for learning known malware behavior and predicting unknown or novel malware. Although anomaly detection is able to detect novel malware, it sometimes causes high false positive. There are several related work applying anomaly detection mechanism to detect mobile malware.

Bose *et al.* [9] present a behavioral detection framework. Instead of the signature-based solutions, they detect mobile malware by observing the logical ordering of an application's actions. Furthermore, they discriminate the malicious behavior of malware from the normal behavior of applications by training a classifier based on Support Vector Machines (SVMs). Schmidt *et al.* [29] extract function calls from binaries of applications, and apply their clustering mechanism, called *Centroid*, for detecting unknown malware. VirusMeter [25] is proposed to detect anomalous behavior on mobile devices based on abnormal power consumption by malware. pBMDS [33] adopts a probabilistic approach through correlating user inputs with system calls to detect anomalous activities in cellphones. It leverages a Hidden Markov Model (HMM) to learn the behavior of the application and the user. Finally, it identifies behavioral differences between malware and human users. Shabtai *et al.* [30], [31] provide a behavior-based Android malware detection named as *Andromaly* to protect the smartphone. They test a series of feature selection approaches for finding the most representative sets of features. *Andromaly* applies several different machine learning algorithms such as *Logistic Regression*, *Bayesian Networks* to classify the collected applications as benign or malicious.

Zhao *et al.* [35] focus on the software behavior based malware detection framework, called *AntiMalDroid*, by using SVM algorithm. The proposed framework dynamically extend malware characteristics into the database.

Teufl *et al.* [32] extract the critical permissions, perform semantic search queries and find relations between obtained permissions for identifying the suspicious anomaly applications.

C. Extracted Features

The extracted features are always helpful and meaningful for constructing analyzed model. In this paper, appropriateness of extracted features affect our emulation results. Therefore, we offer the surveyed features in this section for realizing the categories and definitions of extracted features. We classify the surveyed features into three categories, *dynamic*, *static* and *extra information*.

(1) **Dynamic**

There are several dynamic features used by related work, we list those dynamic features as follows:

- **Logged behavior sequence**: Zhao *et al.* [35] propose *AntiMalDroid* to detect Android malware that use logged behavior sequence as the feature, and construct the models for further detecting malware and its variants effectively in runtime. They also extend malware features database dynamically.
- **System calls**: Burguera *et al.* propose *Crowdroid* that traces the system calls behavior, converts them into feature vectors, and applies k-means algorithm for detecting malware.

– **Dynamic Tainting Data flow and Control flow**: Enck *et al.* [14] propose *TaintDroid*, an extension to the Android mobile-phone platform that tracks the flow of privacy sensitive data through third-party applications. Gilbert *et al.* [20] propose *AppInspector*, that traces both explicit and implicit data flow and control flow for finding security violations.
– **Power consumption**: [13], [22], [25] consider the power consumption as the distinguishable feature between benign and malicious applications.

Shabtai *et al.* [30], [31] propose a Host-based malware detection System that continuously monitors various features and events obtained from the mobile device and then applies machine learning anomaly detectors to classify the collected data as normal (benign) or abnormal (malicious). respectively in 2010 [30] and 2012 [31]. The features they consider including *CPU consumption*, *Number of sent packets through the Wi-Fi*, *Number of running processes*, *Keyboard/Touch-screen pressing* and *Application start-up*.

Furthermore, we also find out other useful features from *DroidBox* or *TaintDroid*. For example, *hashes for the analyzed package* (including MD5, SHA1 and SHA256), *reading and writing operations*, *API calls* to perform encrypted activities, *incoming and outgoing network information*, *loaded classes* through DexClassLoader, *broadcast receivers*, *activated services*, *enforced permissions*, *bypassed permissions*, *information leakage via the network*, *SMS sending* and *phone calls* that all can be expected from *DroidBox*.

(2) **Static**

Another popular method for analyzing the malware is static analysis. It can achieve the goal by analyzing the contents of each applications. Although understanding malware using dynamic analysis can provide a comprehensive view, it is still subjected to high cost in environment deployment and manual efforts in investigation. The static analysis mechanism can reduce the cost and improve the performance. We list several static features used by some related work as follows:

– **Requested permission**: Barrera *et al.* [8] propose a methodology for identifying application clusters based on requested permissions, they try to realize which permission is usually used by specific application category. DiCerbo *et al.* [12] aim at the detection of suspicious applications by using Android security permissions. Kim *et al.* [23] analyze permissions based on DEX and Manifest parsing on malicious applications. Zhou *et al.* [37] detect the infection from know Android malware by permission-based behavior footprinting.
– **Imported package**: Zhou *et al.* [37] also consider packages imported by applications.
– **API calls**: Kim *et al.* [23] and Zhou *et al.* [37] also consider API calls in their works.
– **Instruction (Opcode)**: Zhou *et al.* [36] consider instructions (opcode) for detecting repackaged smartphone applications in third-party android marketplaces.
– **Data flow**: Fuch *et al.* [19] propose *ScanDroid* for automated security certification of Android applications. It analyze data policies in application manifest and data flows across content providers.
– **Control flow**: Chin *et al.* [10] propose *ComDroid* for detecting application communication vulnerabilities that considers the control flow. Grace *et al.* [21] try to detect capability leaks in stock Android smartphones. They build control-flow graph to locate possible execution paths, find the path not protected by the appropriate checks and its entry point is publicly accessible.

(3) **Extra Information**

After the above description, we can realize the dynamic and static analysis are two main mechanisms that analyzes no matter PC or smartphone malware. However, not only the information from the application itself is related and important, but also the information from other aspects.

Zhou *et al.* [36] also consider *application's author* information since they think that it may help to distinguish if the repackaged application is the updated version of original legitimate one or the one being injected some additional functionalities such as the advertisement or malicious patterns.

Teufl *et al.* [32] also consider the application's information displayed on the Android market such as *application description*, *number of downloads*, *price* and *application category* for constructing the *Activation Patterns* proposed by them.

III. Static Functional Behavior Analysis for Android Malware Detection

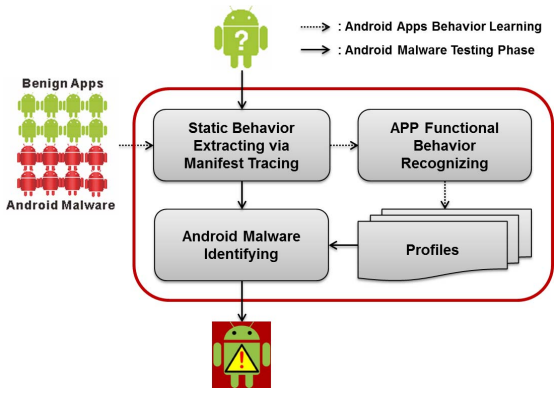

Fig. 1. *DroidMat* Architecture

In this section, we propose a static feature-based mechanism to provide a static analyst paradigm for Android malware

detection with the consideration of detection effectiveness and cross-version capable analysis. The mechanism considers the static information including permissions, deployment of components, Intent messages passing and API calls for characterizing the Android applications behavior. In order to recognize different intentions of Android malware, different kinds of clustering algorithms can be applied to enhance the malware modeling capability. The system architecture is shown in Fig. 1. We describe our approach in the aspect of the architecture and the detail mechanism as following subsections.

A. Observations

Before illustrating what kind of features we use, we first describe some of the observations among benign applications and Android malware. We make a statistics as Table I which count the average usage of each component inside the manifest file between benign applications and Android malware we collected. We can observe that Android malware usually requests more permissions, service and receiver components than benign applications. It implies that Android malware needs more permissions to perform sensitive actions, and as our observations, they usually hide the malicious behavior through running at the background services silently.

TABLE I
THE AVERAGE NUMBER OF PERMISSION AND EACH COMPONENT USED BY BENIGN APPLICATIONS AND ANDROID MALWARE. (UNIT: AVERAGE TIMES)

Component Average Usage

Component Name	Benign Apps	Android malware
`Permission`	4.67	11.27
`Activity`	9.97	3.60
`Service`	0.49	1.10
`Receiver`	0.68	1.73
`Provider`	0.10	0.17

B. Static Behavior Extraction

In our system, we extract the full information about the application including its manifest file and disassembly codes (smali files) for further analysis (apktool [2]). Furthermore, we define the following static features that imply important messages of each application's configurations and intentions. After the feature extraction pre-processing, we construct them as a vector for each application, the value for each element in the vector is 0 if the element is not included in the application while the value is 1 if the element is included in the application.

- First, the *app-specific manifest file* presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. It provides full of information about the application settings. Consequently, there are some distinguishable characteristics such as **(1) Permission** and **(2) Component (including Activity, Service and Receiver)** that we have interest in.
- Second, **Intent** information is also one of features we use. An intent is an abstract description of an operation to be performed. Three of the core components of an application - activities, services and broadcast receivers are activated through intents. With the intent information, we can know what action may be taken.
- Third, the application bytecode contains rich of semantic information such as API calls. We use **API calls** as features to know *what operations the application want to execute* that may be sensitive. Particularly, we focus on those permission-protected APIs from the permission map [16]. In addition, we not only consider those API calls but **their usage in what kind of components**, since API used in different components may imply different intentions. For example, from our observation, Android malware usually collect smartphones information (e.g., IMEI) by calling *getDeviceId() API* in service component that silently run at background, while benign applications calling such API in activity component that show on the user interface. With API calls information, we can infer *intentions* of the app indirectly.
- Fourth, we have surveyed some well-known Android malware such as *ADRD*, *DroidDreamLight* and *DroidKungFu*. We find that most of Android malware like to hidden themselves as services running at background silently and doing some malicious behavior such as stealing private personal information or controlling the device as a bot. Usually, they use receivers to listen and capture the specific broadcasts like *android.intent.action.BOOT_COMPLETE* that would be sent when the Android system finishes booting process. After that, the receiver either handles by itself or starts a service to permanently run at background. The relation between such **Inter-Component Communications** is also what we would like to find. We set components (Activity, Service, Receiver) as entry points and drill down for tracing *ICC* and *API Calls* related to permissions. Finally, the relations would be modeled for further prediction use.

C. Functional Behavior Recognizing

In order to recognize different intentions of Android malware, different kinds of clustering mechanisms can be applied to our system. We choose two of the most common used clustering algorithms, K-means and EM algorithm since their characteristics of simplicity. In this stage, the first encountered issue is to decide the number of clusters. We employ *Singular Value Decomposition (SVD)* method to determine the best number of clusters on the low rank approximation. We notice that there are some other works [24], [26], [27] also apply SVD for deciding the number of clusters. With the help of SVD, we can realize how many information directions (i.e. characteristic with significant differences) from the data.

For benign applications, each of them belongs to a subject category according to its functionality determined by the Android market. We refer to categories on the GooglePlay

and regard each category as a cluster since we assume that applications in the same category may have similar functionalities.

a) Singular Value Decomposition for Intrinsic Behavior Finding: Let IPT be the data matrix representing m Android sources (rows) with n attributes (columns). We decompose IPT as a linear mixture of l bases (row of the base matrix B), with the weights in the columns of the hidden matrix H, using appropriate clustering algorithms. In the system setting, the column size of each Android feature is large that the row size in IPT and would impose the decomposing computation of singular value decomposition (SVD) to extract the critical features. We do not directly use IPT to perform the SVD; in the other words, we perform the step shown in Equation 1.

$$\begin{aligned} IPT \times IPT^T &= U \times D^2 \times U^t \\ U &= \frac{IPT \times IPT^T}{D^2 \times U^t} \\ V &= D^{-1} \times U^t \times IPT \end{aligned} \tag{1}$$

In *DroidMat*, the number of hidden variables should be determined as a priori. In this study, we use the spectrum analysis using SVD and find the number of top singular values l that cover 95% of the total spectral energy. Therefore, we further decompose the IPT using SVD as shown in Equation 2.

$$M = U \times S \times V^t \tag{2}$$

In Equation 2, we use the diagonal matrix S and select the top l concepts that cover 95% of total spectral energy as the number of hidden variables for Equation 3.

$$S = diag(s_1, s_2, ..., s_n) \tag{3}$$

Let k be the smallest integer where λ is the energy spectrum coverage threshold as determined by the analyzer and as shown in Equation 4.

$$\frac{sum(s_1^2 + s_2^2 + ... + s_k^2)}{sum(s_1^2 + s_2^2 + .. + s_n^2)} \geq \lambda \tag{4}$$

D. Android Malware Identifying

The behavior recognition can summarize different functionalities of Android malware for further prediction reference. Applications with similar functionalities may be grouped together. After recognizing applications' behavior intentions, the last goal is to identify the Android malware. Given profiles that represent the characteristics of each cluster, in principle we can apply any classifiers for identifying Android malware. Here, we adopt kNN (with k=1) and NaïveBayes for our evaluation.

IV. Experiment Result

In order to evaluate the proposed approach, we make experiments for evaluating the effectiveness of *DroidMat*. In addition, we compare *DroidMat*with one of well-known Android malware detection tools, called *Androguard*, a signature-based tool [1], [11], published at Blackhat2011 [3]. The applied data for our experiments are introduced in Sec IV-A. The evaluation metrics are described in Sec IV-B. The effectiveness evaluation of proposed approach is in Sec IV-C. Finally, we have experiment discussion in Sec IV-D.

A. Datasets

We collect Android malware from "Contagio mobile" [4] site that seems the only one Android malware public dataset so far, Felt et al. [17] and Zhou et al. [37] also work on the same data source. For benign applications, we randomly downloaded 50 applications from each category on the GooglePlay official Android market and verify them through VirusTotal to ensure at least no anti-virus product recognizing them as Android malware.

We have totally collected 238 Android malware including 34 malware families and 1500 benign applications including 30 categories. Table. II shows the description of our dataset, and Table. III shows Android malware families we use.

TABLE II
THE DATASET DESCRIPTION

	Number of applications	Duration
Benign applications	1500	2012.02
Android malware	238	2011 2012.02

B. Evaluation Metrics

Our evaluation regards each Android malware as an instance. We identify Android malware as true instance and benign application as the false instance. To evaluate the effectiveness of proposed method, we use the confusion matrix to measure *accuracy*, *recall rate*, *false positive*, *false negative*, *precision rate* and *F-measure* in our experiments. The definitions of those metrics are listed as follows:

Here, let TP (true positive) be the number Android malware that are correctly detected; FN (false negative) be the number of Android malware that are not detected (predicted as benign application); let TN (true negative) be the number of benign applications that are correctly classified; and let FP (false positive) be the number of benign applications that are incorrectly detected as Android malware.

- The *accuracy* is defined by

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

- The *recall (true positive) rate* is defined by

$$\text{Recall rate} = \frac{TP}{TP + FN} \tag{6}$$

- The *false positive rate* is defined by

$$\text{False positive rate} = \frac{FP}{TN + FP} \tag{7}$$

- The *false negative rate* is defined by

$$\text{False negative rate} = \frac{FN}{TP + FN} \tag{8}$$

- The *precision* is defined by

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

- The *F-measure* is defined by

$$\text{F-measure} = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{10}$$

C. Effectiveness Analysis

In order to evaluate the effectiveness of *DroidMat*, we setup two series of experiments, one is the baseline comparison, and the other one is to compare with Androgaurd, one of the popular open source of Android malware analysis and detection.

1) Effectiveness of The Baseline Comparison: In this subsection, we make the experiment to realize the effectiveness of *DroidMat* that applying different baselines (i,e,. feature set, clustering and classification algorithms). Fig. 2 shows the experiment result of applying different feature sets to *DroidMat*.

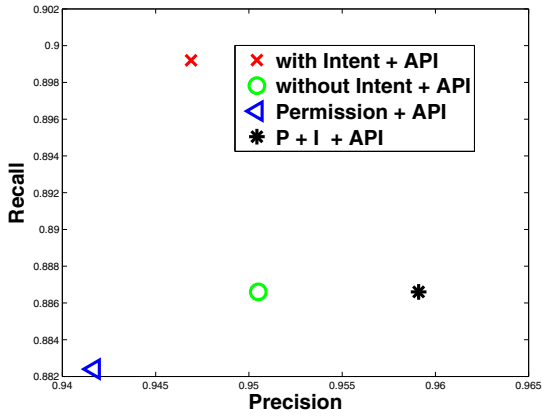

Fig. 2. Android application features comparison from the precision-recall plots.

We list the explanations of each feature set as follows:

- **With Intent + API**
  This feature set include all of information about the application, including *component classes*, *requested permissions*, *intents* and *API calls*.
- **Without Intent + API**
  This feature set is almost the same as previous one but without considering the *intents* information.
- **Permission + API**
  This feature set include *requested permissions* and *API calls*.
- **P + I + API (The feature set used in our approach)**
  This feature set is used by our approach that include *requested permissions*, *intents* and *API calls*

The result of baseline effectiveness comparison shows that the feature set, *"with Intent + API"* has the best result from the *recall rate* point of view. And the feature set, *"P + I + API"*, used in our approach, has the best result from the *precision* point of view. Although the former consists of whole information related to applications that can detect more Android malware, it might cause the overfitting problem since the part of feature set include *component class names* that are unique to each different application. Furthermore, although the recall rate of the feature set used in our approach is not as good as the one that contains whole information about the applications, our feature set has the best result from the *precision* point of view and may not cause the overfitting problem since those features are more general, not specific to each or small set of applications.

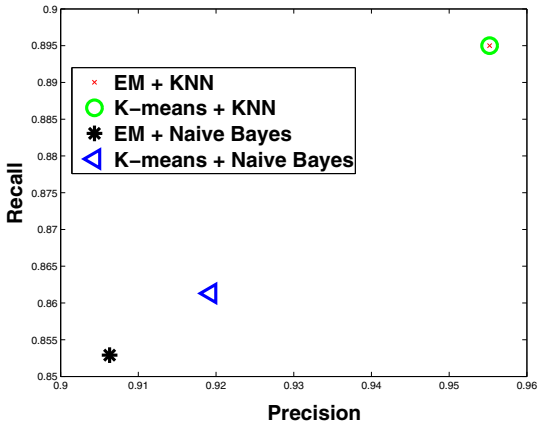

Fig. 3. Android malware detection analysis in different baseline algorithms comparison from the Precision-Recall view.

Fig. 3 shows the experiment result of applying different clustering and classifying algorithms to our approach. Four different combinations of clustering and classifying algorithms are used for analysis.

We found that *k-means* has both the best recall and precision from the clustering point of view. *kNN* (with k=1) has the best result from the classification point of view. We finally choose the combination of k-means and kNN as our best result. After we analyze that why k-means and kNN has the best result, we found that since the features we select are distinguishable to Android malware from benign applications.

2) Effectiveness Comparison with Androgaurd: Finally, we make the effectiveness comparison of Android malware detection capability between *DroidMat* and Androguard. Table III shows the experiment result. Furthermore, the confusion matrix and the evaluation metrics comparison of both system are shown in the Table IV and Table V respectively.

The experiment result shows that *DroidMat* can detect more Android malware than Androguard. Even in the case of very few samples (only two or three samples), *DroidMat* can also have the good result in detecting Android malware. Furthermore, *DroidMat* can identify most variants of Android malware. For example, in some variants of Android malware such as *FakeInstaller*, *FakeNotify*, *FakePlayer* and *GoneSixty* that are difficult to be identified by Androguard while *DroidMat* can identify them. From the average of recall rate and F-measure, the *DroidMat* perform over about 38% recall rate

and 25% F-measure than Androguard respectively.

TABLE III
DETECTION EFFECTIVENESS BETWEEN *DroidMat* AND ANDROGUARD, THE COLUMN VALUE WITH PERCENTAGE IS RECALL RATE, AND THE UNIT FOR SAMPLES IS PIECE

Malware Family	Androguard	*DroidMat*	Samples
ADRD	100.00%	0.00%	1
AdSMS	0.00%	100.00%	1
AnserverBot	0.00%	100.00%	1
Arspam	0.00%	0.00%	1
BaseBridge	55.56%	44.44%	9
FakeInstaller	70.59%	88.24%	17
CarrielQ!Android	0.00%	0.00%	2
DroidDreamLight	100.00%	100.00%	4
DroidDream	100.00%	100.00%	1
DroidKungFu	80.00%	40.00%	5
FakeNotify	0.00%	100.00%	89
FakeBattScar	0.00%	0.00%	1
FikeNefix	0.00%	0.00%	1
FakePalyer	0.00%	100.00%	3
FakeTimer	0.00%	0.00%	1
FlexiSpy	0.00%	100.00%	1
Foncy	50.00%	100.00%	2
Geinimi	100.00%	100.00%	23
GGTracker	100.00%	50.00%	2
GoldDream	100.00%	100.00%	2
GoneSixty	0.00%	100.00%	5
HippoSms	100.00%	0.00%	1
Kmin	100.00%	100.00%	38
Lotoor	66.67%	83.33%	6
Lovetrap	100.00%	0.00%	1
Pjapps	100.00%	100.00%	11
RogueSPPush	100.00%	0.00%	1
RootSmart	0.00%	0.00%	1
SmsHider	100.00%	0.00%	1
Spitmo	100.00%	0.00%	1
Tapsnake	100.00%	0.00%	1
Walkinwat	100.00%	0.00%	1
YZHCSMS	100.00%	100.00%	2
Zitmo	100.00%	0.00%	1
Average	49.58%	87.39%	238

TABLE IV
THE CONFUSION MATRIX OF ANDROGUARD AND *DroidMat*

(a) The confusion matrix of Androguard

Predicted as –>	**Benign**	**Malicious**
Benign	1499	1
Malicious	120	118

(b) The confusion matrix of *DroidMat*

Predicted as –>	**Benign**	**Malicious**
Benign	1493	6
Malicious	30	208

TABLE V
THE COMPARISON OF EVALUATION METRICS

Target	**Accuracy**	**Recall**	**Precision**	**F-measure**
Androguard	0.9304	0.4958	**0.9916**	0.6611
DroidMat	**0.9787**	0.8739	0.9674	**0.9183**

D. Experiment Discussion

Due to considering *permissions*, *intents* and *API calls* including their usage in what kind of components in the feature-based mechanism that are highly related to the Android application, the experiment result shows that *DroidMat* can detect more Android malware than Androguard. However, the number of Android malware families detected by *DroidMat* is a little less than Androguard since most of them are only with a single sample which limits *DroidMat* to learn their behavior well. Although *DroidMat* has the limitation on detecting the Android malware only with a single sample, it still can perform well in the case of the Android malware with more than two samples. There are two Android malware families, *BaseBridge* and *DroidKungFu*, which *DroidMat* can't perform well on detecting them. We discuss the characteristics of them respectively as follows:

- **BaseBridge**
  According dissecting Android malware by Zhou *et al.* [34], they mentioned that when a *BaseBridge*-infected application runs, it will check whether an update dialogue need to be displayed. If yes, the user will be prompt that a new version is available. If the user accepts, an *"updated"* version with actual malicious payload will then be installed.
- **DroidKungFu**
  The *DroidKungFu* malware is similar to *BaseBridge*. But instead of carrying or enclosing the *"update"* version inside the original application, it remotely download a new version from network. Moreover, it notifies the users through a third-party library that provides the legitimate notification functionality.

According to the previous discussion, because both *BaseBridge* and *DroidKungFu* Android malware extract the actual malicious payload from external places rather the original applications themselves, *DroidMat* with the static-based mechanism can't correctly detect them.

V. CONCLUSION

We proposed *DroidMat*, a novel approach to distinguish and detect Android malware with different intentions. *DroidMat* has the following properties. *Effectiveness*: It is effective, that is, it is able to distinguish variant of Android malware between distinct purposes of them. It achieves up to 97.87 percentage points in accuracy. *Scalability*: It is scalable, that is, it is linear in the size of the problem (i.e., the number of non-zeros in the input matrix). *Efficiency*: It does not need to dynamically investigate the Android application behavior from the sandbox or by emulation, which saves the cost in environment deployment and manual efforts in investigation.

ACKNOWLEDGEMENT

This research is supported in part by the National Science Council of Taiwan under grants number NSC100-2218-E-011-009.

REFERENCES

[1] Androguard. http://code.google.com/p/androguard/.

[2] Android-apktool. http://code.google.com/p/android-apktool/.

[3] Black hat abu dhabi 2011. http://www.blackhat.com/html/bh-ad-11/bh-ad-11-speaker_bios.html.

[4] Contagio mobile. http://contagiominidump.blogspot.com/.

[5] Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth. http://www.gartner.com/it/page.jsp?id=1924314.

[6] Smartphone and app growth soars: Infographic. http://blog.mylookout.com/blog/2012/02/06/smartphone-and-app-growth-soars-infographic/.

[7] Lookout app genome report. https://www.mylookout.com/appgenome, 2011.

[8] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM.

[9] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 225–238, New York, NY, USA, 2008. ACM.

[10] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 239–252, New York, NY, USA, 2011. ACM.

[11] A. Desnos and G. Gueguen. Android: From reversing to decompilation. Blackhat, 2011.

[12] F. Di Cerbo, A. Girardello, F. Michahelles, and S. Voronkova. Detection of malicious applications on android os. In *Proceedings of the 4th international conference on Computational forensics*, IWCF'10, pages 138–149, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] B. Dixon, Y. Jiang, A. Jaiantilal, and S. Mishra. Location based power analysis to detect malicious code in smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 27–32, New York, NY, USA, 2011. ACM.

[14] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[15] 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, pages 235–245, New York, NY, USA, 2009. ACM.

[16] 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, pages 627–638, New York, NY, USA, 2011. ACM.

[17] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 3–14, New York, NY, USA, 2011. ACM.

[18] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[19] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. SCanDroid: Automated Security Certification of Android Applications. Technical Report CS-TR-4991, Department of Computer Science, University of Maryland, College Park, November 2009.

[20] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung. Vision: Automated security validation of mobile apps at app markets. In *Proceedings of the second international workshop on Mobile cloud computing and services*, MCS '11, pages 21–26, New York, NY, USA, 2011. ACM.

[21] M. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic detection of capability leaks in stock Android smartphones. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, Feb. 2012.

[22] H. Kim, J. Smith, and K. G. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, MobiSys '08, pages 239–252, New York, NY, USA, 2008. ACM.

[23] S. Kim, J. I. Cho, H. W. Myeong, and D. H. Lee. A study on static analysis model of mobile application for privacy protection. In J. J. (Jong Hyuk) Park, H.-C. Chao, M. S. Obaidat, and J. Kim, editors, *Computer Science and Convergence*, volume 114 of *Lecture Notes in Electrical Engineering*, pages 529–540. Springer Netherlands, 2012. 10.1007/978-94-007-2792-2_50.

[24] S. Lee and M. Hayes. Properties of the singular value decomposition for efficient data clustering. 11(11):862–866, November 2004.

[25] L. Liu, G. Yan, X. Zhang, and S. Chen. Virusmeter: Preventing your cellphone from spies. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 244–264, Berlin, Heidelberg, 2009. Springer-Verlag.

[26] P. Liu, S. X. D. Tan, H. Li, Z. Qi, J. Kong, B. McGaughy, and L. He. An efficient method for terminal reduction of interconnect circuits considering delay variations. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, ICCAD '05, pages 821–826, Washington, DC, USA, 2005. IEEE Computer Society.

[27] P. Liu, S. X.-D. Tan, B. McGaughy, L. Wu, and L. He. Termmerg: An efficient terminal-reduction method for interconnect circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(8):1382–1392, 2007.

[28] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically rich application-centric security in android. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 340–349, Washington, DC, USA, 2009. IEEE Computer Society.

[29] A.-D. Schmidt, J. H. Clausen, A. Camtepe, and S. Albayrak. Detecting symbian os malware through static function call analysis. Number March 2006, pages 15–22. IEEE, 2009.

[30] A. Shabtai and Y. Elovici. Applying behavioral detection on android-based devices. In *MOBILWARE*, pages 235–249, 2010.

[31] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. "andromaly": A behavioral malware detection framework for android devices. *J. Intell. Inf. Syst.*, 38(1):161–190, 2012.

[32] P. Teufl, S. Kraxberger, C. Orthacker, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber. Android market analysis with activation patterns. In *MOBISEC*, 2011.

[33] L. Xie, X. Zhang, J.-P. Seifert, and S. Zhu. pbmds: A behavior-based malware detection system for cellphone devices. In *Proceedings of the third ACM conference on Wireless network security*, WiSec '10, pages 37–48, New York, NY, USA, 2010. ACM.

[34] X. J. Yajin Zhou. Dissecting android malware: Characterization and evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, May 2012.

[35] M. Zhao, F. Ge, T. Zhang, and Z. Yuan. Antimaldroid: An efficient svm-based malware detection framework for android. In C. Liu, J. Chang, and A. Yang, editors, *ICICA (1)*, volume 243 of *Communications in Computer and Information Science*, pages 158–166. Springer, 2011.

[36] 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.

[37] 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 19th Annual Network & Distributed System Security Symposium*, Feb. 2012.