Screening Smartphone Applications Using Behavioral Signatures

Suyeon Lee, Jehyun Lee, and Heejo Lee*

Division of Computer and Communication Engineering,
Korea University,
Seoul, Korea
{suyeonl,arondit,heejo}@korea.ac.kr

Abstract. The sharp increase of smartphone malwares has become one of the most serious security problems. The most significant part of the growth is the variants of existing malwares. A legacy approach for malware, the signature matching, is efficient in temporal dimension, but it is not practical because of its lack of robustness against the variants. A counter approach, the behavior analysis to handle the variant issue, takes too much time and resources. We propose a variant detection mechanism using runtime semantic signature. Our key idea is to reduce the control and data flow analysis overhead by using binary patterns for the control and data flow of critical actions as a signature. The flow information is a significant part of behavior analysis but takes high analysis overhead. In contrast to the previous behavioral signatures, the runtime semantic signature has higher family classification accuracy without the flow analysis overhead, because the binary patterns of flow parts is hardly shared by the out of family members. Using the proposed signature, we detect the new variants of known malwares by static matching efficiently and accurately. We evaluated our mechanism with 1,759 randomly collected real-world Android applications including 79 variants of 4 malware families. As the experimental result, our mechanism showed 99.89% of accuracy on variant detection. We also showed that the mechanism has a linear time complexity as the number of target applications. It is fully practical and advanced performance than the previous works in both of accuracy and efficiency.

Keywords: Smartphone security, Android, Malware, Runtime semantic signature.

1 Introduction

Smart devices are now facing a serious threat posed by surging malwares. Smartphone has become the most popular target for malware writers since it contains a great deal of user information and has capability for mobile billing. The majority of smartphone malwares leak user information and perform user unwanted billing

* Corresponding author.

L.J. Janczewski, H.B. Wolfe, and S. Shenoi (Eds.): SEC 2013, IFIP AICT 405, pp. 14–27, 2013.

by abusing smartphone's original functionality. Recently, smartphone malwares have adopted several obfuscation techniques such as metamorphism to avoid detection. According to F-Secure's report [1], about 82% of newly discovered mobile malware are revealed as a variant of known malware family. Such trend is especially remarkable on *Google Android*. Overall, smartphone malwares can cause more direct invasion of privacy and credential damage to the users compared with the desktop malwares. However, the flood of metamorphic malwares on the smartphones impedes an efficient dealing of malware attack. Accordingly, a mechanism which prevents malwares by filtering variant of known malwares, efficiently, is needed to retain smartphone security and user privacy.

Previous approaches for variant detection based on behavior analysis were not appropriate to identify the malware family where a detected malware variant belongs to. Those approaches detect the variants by estimating similarity of behavior such as API call frequency or API call sequence [2], with a known malware. Extracting and comparing the behaviors from numbers of target executables takes heavy computing overhead. Detection based on similarity of behavior is a helpful way for unknown detection, but it does not provide and use any evidence to show that certain variants are derived from same malware.

In opposite, a representative signature that usually has been used by AV vendors is effective to define and detect malware family. It is also efficient in contrast to the behavior based approaches in terms of time and space complexity. However, the signatures have not only narrow detection coverage on a malware family due to the lack of semantic but also easily defeated by the malwares which are adopted code obfuscation such as metamorphism. As a conclusion, re-investigation of overall code area for behavior analysis and to make an additional signature for slight modulation of a malware is an inefficient way against the little effort that consumed for making a variant.

In this paper, we propose a variant detection mechanism which filters new variants of known malware family. Since the most Android malwares are repackaged version of legitimate executable file, the malwares in same family retain semantics unchanged. Using this feature, we detect the variants efficiently and accurately by analyzing such semantics in static. The proposed mechanism uses a runtime semantic signature of known malwares. The runtime semantic signature is a malware family signature including the family representative binary patterns for control and data flow instructions and character strings, as well as API calls. The signatures including flow instructions and family representative signatures contribute to achieve accurate variant detection and family classification reducing analysis overhead.

In experimental evaluation, our mechanism show high detection performance and consumes practically low time in variant detection. We evaluated our mechanism with 1,759 real-world Android applications including 79 variants of 4 malware families, *DroidDream*, *Geinimi*, *KMIN*, and *PjApps*. The experimental set is randomly collected by automated crawler during the period of September 2011-December 2011. For performance evaluation, first, we created runtime semantic signatures for 4 malware families. Our mechanism showed over 99% of

detection accuracy and near 97% of recall performance, on average, from 10-fold cross validation. Second, for unknown detection performance, we compared the four family signatures with unidentified 100 malwares. Our mechanism detected 56 malwares from the experiment set. This result shows that our mechanism detects code-level invariants with same semantics, while legacy signature-based approaches are generally not able to detect such variants. Finally, in the scalability evaluation, our mechanism screens a thousand of applications with also a thousand of signatures within 23 seconds. To consider efficiency and accuracy against variant detection of our mechanism, it is applicable for screening Android applications before they are uploaded on public app markets.

Our contributions are two folds:

- We proposed a runtime semantic signature that is used for detecting variants of known malware family accurately and efficiently. The runtime signature is a signature for a malware family sharing its API calls and representative part of codes having control and data flow semantics. It solves an existing malware detection issue, by combining malware semantic with sequence information. This contribution makes it possible to detect malware including their variants, even if the variant adopted evasion technique such as metamorphism.
- We reduced the number of signatures. The runtime signature representing a malware family on a single set of signature and covers numbers of family members including newly appeared variants. The runtime signature is based on the sequence of API calls which are shared among the malwares which have similar behaviors, but the adaptation of family representative binary patterns enable to detect and to classify malware families in practical accuracy. It enables to efficiently respond to the exponentially increasing number of malwares.

The rest of this paper organized as follows: we will start from describe details of mechanisms and assumptions of our proposition in section 2. Next, we will present experimental result of our system on Android in section 3. After distinguishing our work with previous works in section 4, we will discuss about limitations, future work and finally conclude our work in section 5.

2 Malicious Application Detection Using Behavioral Signature

2.1 Mechanism Overview

Our detection mechanism is an advanced type of signature-based approach. On a smartphone, the number of target of inspection (i.e. applications) are increasing sheerly and malware variants are taking the majority share of new malwares.

Legacy signature-based approach scans a lot of applications in a timely manner. However, since the performance of signature-based approach is highly dependent on its signature, it is not robust against a number of malware variants. Therefore, we added runtime semantic to complement the weakness of signature-based approach.

To detect malware variants belonging to a family by a single robust signature set, we basically use the malicious behaviors shared by family members represented by API calls and control and data flow between the calls. The API call sequence is one of the well-known behavior based approaches for detecting malware variants and reducing the volume of signatures. However, it has false detection problem in the smartphone environment because the legitimate smartphone applications have much more similar behavior to the malicious applications in contrast to the legacy PC environment's. Due to the problem, in the smartphone environment, the variant detection should be performed with more critical evidences representing the membership of a family. We overcome this challenge by adapting binary patterns of instructions between API calls for control and data processing to the behavioral signature. In the legacy PC environments, the binary patterns of control and data processing have too many variations and not so useful due to the numbers of various APIs. However, the executables working on a smartphone have relatively small variety of APIs and instructions enough to use as a behavioral signature. The binary patterns of the control and data flow, the runtime semantics, are general enough to detect the variants of a malware generated by code and class reusing or repackaging but rarely shared to the other family members and benign applications. The proposed detection mechanism using the behavioral signatures shows practical enough detection performance in both of efficiency and accuracy in our evaluation.

The key features of proposed behavioral signature structure are two-folds. First, the signature contains binary patterns of API call instructions on an executable file. In case of the Android environment, an application has its executable code as a *Dalvik Executable(DEX)* file. Second, the signature contains runtime semantic for reducing control and data flow analysis time and classifying malware families. Runtime semantic is also bytecode patterns that are used for the data and control flow between API calls. While analyzing known malwares, we monitor the taint flow of sensitive APIs and associate flow between APIs with three relationship, flow, call and condition. Figure 1 shows the overall architecture of our proposed mechanism. To efficiently and reliably winnow new malwares from target applications, our mechanism conducts two phases of analysis. Each analysis phases are quite straightforward. We first construct the behavioral signatures based on the known malware binaries and analysis report of them. Then we match the signatures with target DEX files. These analysis phases consequently produce a set of similarities between signatures and target applications as well as the security report of target applications. In the remainder of this section, we will detail each phases.

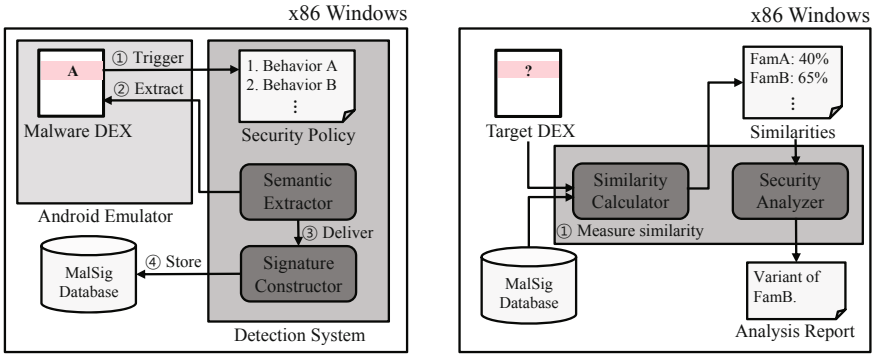

Fig. 1. Overview of the proposed variant detection mechanism

2.2 Signature Construction

In this section, we explain the proposed behavior signature from the definition of malicious behavior. Signature construction is started from detecting the malicious behaviors by dynamic analysis. The signature construction process extracts the binary patterns and character strings from known malwares and estimates the weight of each pattern and string depending on the family where they belong to.

Malicious Behavior Definition. Before scanning applications, we need to clarify what behaviors will be considered as malicious behaviors. According to the 'Malicious payload' classification of Y. Zhou *et al.* [3], we selected four severe behaviors for malware detection. We detailed the definition of each behavior below.

- **Privilege Escalation** Since Android platform consists of more than 90 open-source libraries as well as linux kernel, flaws included in such libraries naturally incur vulnerabilities of whole Android system. As the time of research, seven exploits have been reported that are possible to gain the root privilege of an Android device. By adopting the exploits, application is being able to perform kernel-level control of the device without any user notification. The most risky and widespread malwares such as *DroidDream* and *DroidKungfu* initially contain exploits to perform high-risk malicious activities surreptitiously. Since the exploitation is the most serious threat for users, we detect known privilege escalation exploits that either contained in known malwares or be searched on internet forums.
- **Remote Control** Over 90% of currently reported Android malwares have remote control capability. Specifically, Android malwares that have remote control capability follow the commands from designated C&C server via

HTTP web request or SMS messages. More recently, malware authors obfuscate the C&C server IP address or the commands to make malware analysis be effortful. We identified three specific behaviors to detect remote control behavior, (1) establishing internet procedure and registering broadcast intent for SMS messages, (2) receiving internet packet or SMS messages, (3) and application or kernel-level runtime execution of received data.

- **Financial Charge** Financial charge is the most profitable way for malware authors. Since SMS messages can be sent surreptitiously i.e. without any user notification on Android, many malwares are designed to send premium-rate SMS or phone call. In early times, malwares have had hard-coded numbers to make charge for users. However, recent malwares with financial charging capability are being more complicated by changing their phone number gaining method such as runtime push-down from C&C server and to get from a encrypted file inside of the package. To detect financial charge capability, we monitor APIs that send messages (e.g. sendTextMessage) to hard-coded number.
- **Information Collection** Since a smartphone is one of the most trusted and user-friendly devices, it contains a great deal of information which are deeply related to the owner's social life and credentials. Therefore, malware authors are trying to get wide range of information from device-specific information (e.g. IMEI, IMSI and phone number) to the owner's information (e.g. contact book, SMS messages, call log and credentials). The exfilteration of such information will affect not only the user oneself but also the people around him/her both directly and indirectly. We identify the APIs that provide such information and network transfer API that will possibly send the information to remote server.

Malicious Behavior Detection. We identify each behavior as a set of APIs. However, defining APIs that form each behavior is difficult because the APIs vary depending on the sort of malware. For efficient and reliable analysis, we dynamically analyze known malware samples and extract APIs that are considered to malicious behaviors based on the corresponding malware reports. Additionally, to classify the family if target application has identified as a malware, we extract characteristics from malware that can represent the whole malware family. Variants of many Android malwares have common strings, constants or even classes or methods in practice. We extract the characteristics that only appear in each family. Common strings and constants between families are not considered as proper characteristics.

Signature Construction. Our signature is devised to provide knowledge-base for further investigations. To achieve reliable and efficient malware detection, we need to design our signature structure to meet four requirements in below. Basically, signatures should contain (1) behavioral semantics for basic detection capability. And, if a target application identified as a variant of known malware, then it could be (2) identifiable as a member of certain malware family. Also, to

overcome the pitfalls of legacy signature-based approach, our signature should be (3) reliable against different evasion techniques while maintaining (4) the efficiency as a signature-based approach.

Figure 2 illustrates the structure of behavioral signature with an example. A signature represents a malware family. In other words, it is capable to know whether the target application is a new member of known malware family with a single matching. The signature consists of three main elements. The first one is malicious APIs and their runtime semantics for control and data flow. We extract APIs that make malicious behavior mentioned in *Malicious Behavior Detection*. When extracting APIs, runtime semantics such as repeated count of the API or the relationship between the former and latter APIs will also be extracted to infuse semantics into signatures. The second one is family characteristics. Since Android applications share broad range of behaviors even they are the member of each different malware families, identifiable information for malware family is necessary. Android malwares which included in same family tend to share same strings, constants, methods or even classes in most cases. Based on this tendency, we extract family common string, constants, methods and classes as family characteristics for family identification. The third one is weights of each behavior within family. Note that the signature contains sets of APIs and semantics according to that signature represents every behaviors appear in every variants of certain family. More frequently used API will take greater weight.

In conclusion, by using signature, our mechanism detects malicious behaviors semantically with APIs and their runtime semantics and also identifies the family of newly detected malware with API weights and family characteristics simultaneously.

In terms of signature matching, since a signature represents one malware family, it is possible to use only a small number of signatures to cover a large number of malwares. Our mechanism scans the signatures with target applications efficiently in conjunction with a static matching algorithm. We will describe the matching algorithm in detail in following section.

2.3 Malware Detection

Similarity Measurement. Similarity calculator compares and estimates the similarity between the DEX file of target application and behavioral signatures. Each signature has a weight of each behavior based on its discernment on malware family identification. Similarity calculator first scan all potentially malicious behaviors that contained in a DEX file, based on the APIs and semantics that stored in each signature. Specifically, the behavior is represented as the name and DEX bytecode patterns of API calls for faster scanning. However, DEXs are different by applications even if they contain same APIs. Thus direct matching bytecode patterns gathered from a known malware and from a target application is implausible. Instead, we separate constant area and variable area from DEX bytecodes. For example, *invoke-virtual (arg1, arg2, arg3, arg4, arg5) methodA* in Dalvik instruction is corresponds to *6e 35c (4bit, 4bit, 4bit, 4bit, 4bit,*

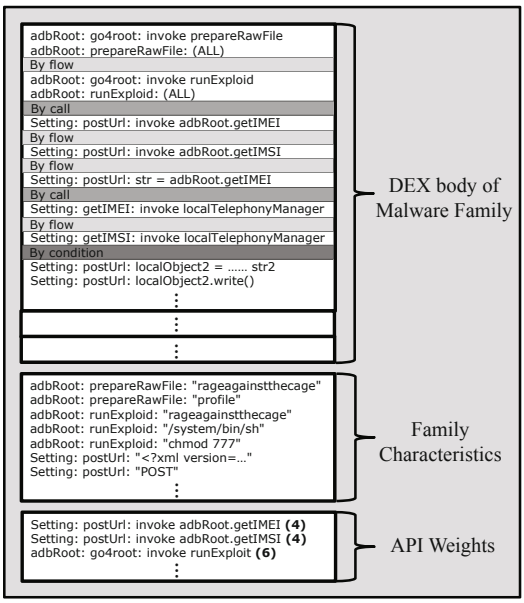

Fig. 2. An example of malware family signature, *DroidDream*

4bit) 16bit in hexadecimal code. In this case, *6e 35c* is constant part that does not differ by application and *(4bit, 4bit, 4bit, 4bit, 4bit, 4bit) 16bit* is variable part that differs by application. And second, similarity calculator estimates the similarity *S(T, A)* between target DEX *'T'* and a signature *'A'*. The similarity measuring is alike as follows:

$$S(T, A) = \frac{\sum_{j=1}^{n} W(b_j \mid b_j \in (T \cap A))}{\sum_{i=1}^{m} W(b_i \mid b_i \in A)}$$

Security Analysis. Security analyzer decides the maliciousness of a target application and discerns the most similar malware family with the target application. The decision method is quite straightforward. Among the resulted similarities for each family signature, the family which has the highest similarity is decided as the original family of target application. If the application has approximately equal similarities with multiple malware family, then the family characteristics are additionally used for more accurate identification.

3 Performance Evaluation

To evaluate the practicality of our malware detection mechanism, we performed time efficiency and detection performance evaluation experimentally.

The experiments are performed on a desktop PC which has 2.8 GHz Intel dual-cores CPU, 2GB RAM and Microsoft Windows XP SP3 as the OS. Our self-developed experimentation program in C++ measures time consumption and detection accuracy on malware variants detection.

3.1 Data Set

For the performance evaluation, we gathered 79 variants on four famous malicious Android applications and 1,680 legitimate applications published in real world. In detail, the variants set consists of 11 variants of *DroidDream*, 12 variants of *Geimini*, 40 of *KMIN* and 16 of *PjApps* variants. The malwares are gathered from public malwares data bases on the Internet. On average, the DEX files of the malware variants have 340 KB of size and the legitimate applications have 260 KB of size.

3.2 Signature Set

Behavioral signatures of known malwares for the detection are constructed from pre-analyzed and published malicious behaviors. We extracted class names, method instruction bodies and internal strings of methods which works for the malicious activities from the DEX files. In our experiments, the signatures which are extracted from randomly chosen training set have approximately 10 KB of size per a malware family. On the other hand, the white signatures which are trained from over a thousand of sample legitimate applications have 3 KB of size.

3.3 Experimental Result

Variant Detection Performance. The proposed system detects and identifies a new malware as a variant of known malware first. The major part of variant detection is similarity calculation. In contrast to the legacy signature matching method, our detection method investigates how much similar a new application is to the known malicious ones. The similarity to a malware family of an application is determined by the ratio of shared signatures. The detailed way for similarity calculation is explained at the *Signature Similarity Calculation* section. If it is determined as an unknown in this step, it means that the target application is the legitimate or a new malware family which is not corresponding to any known malware families. The application needs to be analyzed at the dynamic analysis phase.

For variant detection and identification performance evaluation, we performed 10-fold cross validation with the real-world malware samples. We made ten groups per a malware family. Then we took one group for signature extraction and rest nine groups as detection targets. We performed the testing ten times with randomly generated group configurations.

The variant detection system shows reliable detection performance. In the best threshold configuration, it shows 99.89% of accuracy and 98.73% of F-measure value on the 10-fold cross-validation results. Even though the volume of legitimate samples are much more than the malware samples, the performance showed on Table 2 is remarkable compared to other previous approaches.

The experimental result illustrated on left side of Figure 3 shows the recall rate more than 90% of even in the detection thresholds higher than 30% of similarity which have no false positive rate. Recall rate is the rate of detecting variants as a corresponding family. Though several variants of *Geimini* and *DroidDream* share some methods and strings and show over 30% of similarity, there are no family-mismatched detection cases. If they share many same malicious functionalities, their signature likely share the same API calls and methods. In these cases, the detection results of malwares belonging to of the similar families have the false negative decisions. At last, analysis for the false detection of legitimate application will be discussed with effect of the white signature.

The malware family which shows the lowest detection performance is *DroidDream*. According to our manual investigation results using decompiling, the major reasons of the lower similarity are the difference of included classes and methods. Several *DroidDream* variants only take rooter and self-alarm event methods within the known malicious functionalities of the *DroidDream*. One possible guessing is that the malwares share just a small part of code with the *DroidDream* such as the popular rooting exploit *rageagainstthecage*.

The detection system is also effectible to detect unknown malware not yet analyzed. Because several malicious actions and their code are commonly utilized on different kinds of malware, the trained signatures enable to detect the functionally similar unknown malwares. Table 1 shows the detection result on one hundred of non-labeled Android malwares using four signature sets and 25% similarity threshold. As the result, 56% of malwares are detected as the variants of four known malware families. It means that the 56% of malwares has the same name, strings, or same methods whose definition is identical to the one of the four malware families. In contrast, the rest 44% of malwares means that they have new methods and classes which are not included to any of the four signatures. In terms of the detection rate, the unknown malware detection rate could be increased to practical level if the knowledge-base had various and many enough signature sets.

Time Efficiency. Our malware detection system has an advantage on its time efficiency. A variant of a known malware whose behavioral signature is in a data base is detected as a variant of the known before a detailed and heavy inspection such as source level analysis or run-time testing on a sand-box. In our implementation for the experiments, the behavioral signature matching uses a matching algorithm using a hash-tree which takes a constant time [4]. As illustrated on Figure 4, the number of signatures has little effect to its time consumption. Consequently, the mechanism shows linearly increasing time consumption along with the amount of target applications. It means that this front line variant

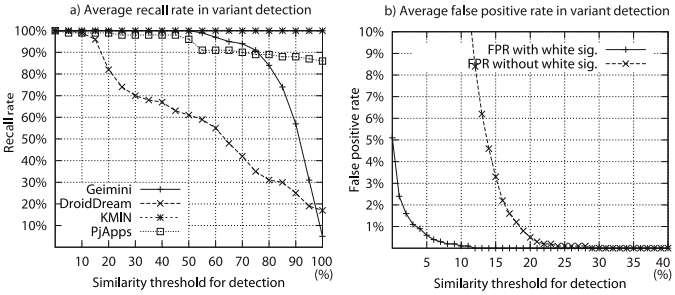

Fig. 3. Average recall(left) and false positive rate(right) on the variants detection experiment

Table 1. Investigation result for non-labeled malwares

DroidDream	Geimini	KMIN	PjApps	Total
4%	23%	29%	0%	56%

detection mechanism reduces the amount of target applications to be analyzed much smaller with a reasonable time overhead.

Effect of White Signature and Redundant Removal. The proposed system reduces the false detection by adopting white signatures. By giving negative points to the applications which have the anti-malicious methods and classes, the detection system avoids false detection of critical but safe applications.

The DEX bytecode patterns and strings which appear in both of malicious and legitimate applications are considered as the redundant. The redundant patterns are removed the signatures and ignored on the detection process. The bytecodes patterns and strings which are only in the legitimate applications and representative for the legitimate applications are considered as a white signature. In contrast to the redundant patterns, a white signature is rarely gathered because a white signature must be only on authenticated legitimate applications but not shared to any malware even though a repackaged malware also has legitimate codes on its DEX file.

For evaluating the effect of white signature and redundant removal, we performed the static variant detection to 1,680 of legitimate Android applications using the signature sets which are used at the variant detection experiment. The right side of Figure 3 shows the effect of adopting white signatures in false positive reduction. Among all the ranges of false positive occurrence, the rates are significantly decreased.

In the comparison with the static analysis study of Schmidt *et al.* [5] on classifying Android executables, our approach shows better performance on the correctly classified instances rate keeping non-false positive rate. In comparison with the study of Shabtai *et al.* [6] applying machine learning using hundreds

Table 2. Detection Performance Comparison Table

Method	Accuracy	Recall	Precision	F-measure
Androguard	93.04%	49.58%	99.16%	66.11%
DroidMat	97.87%	87.39%	96.74%	91.83%
Proposed	99.89%	97.73%	99.74%	98.73%

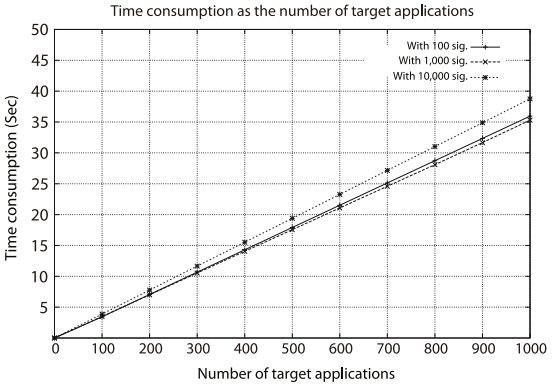

Fig. 4. Time consumption for detection as the number of target applications and signatures

of features extracted from DEX and XML, our classifying result shows better performance than the accuracy of their best configuration.

4 Related Work

The previous work for Android malware is mainly focused on the behavior and trainable features of source code and executable files. The studies which take the machine learning approach [6,7] attempt to classify the malware from the legitimate applications using characteristic features of malware. The classifying approaches using machine learning are robust to the small changes on malware variants. However, the malware has much different behavior and capability along with their family, and the result of these works are hard to give information and detection evidences. Furthermore, the base legitimate applications significantly affect their function and API call statistics. In contrast, our approach classify the malware into each malware family and it gives the detailed information about their behavior.

The behavior analysis approaches are classified into static approaches [2,5,8] and dynamic approaches [9,10]. The static approaches not limited on the behavior based approaches are light-weight and scalable. However, they have a limitation on the accuracy because it is hard to tracking the exact behavior even though the target application can be decompiled. In contrast, the dynamic analysis approaches using taint analysis and API monitoring have ability to tracking

the behavior accurately on run-time. But the dynamic analysis approaches have the efficiency problem because of the requirement of time and resources including a virtual environment and test execution. In terms of efficiency, our work takes a matching approach which is fast as the static analysis and even more efficient against the numbers of variants using the behavioral signature. And the behavioral signature we proposed proves pre-investigated behaviors with exact evidences.

5 Conclusion

In this paper, we proposed a scalable and accurate co-operated approach for Android malware detection. The proposed system overcomes the trade-off problem between the efficiency and accuracy. The proposed system solves the efficiency problem of the dynamic analysis approach due to the virtual environment and test execution by adopting static analysis approach using signatures which are faster and lighter. And the accuracy problem of the static analysis caused by the lack of robustness is solved by using a behavioral signature. The proposed behavioral signature, the runtime semantic signature, includes binary patterns for entity names and instructions for control and data flow over the legacy API calls for malicious acitvities. We experimentally showed that the runtime semantic signature improves the accuracy compared with the previous static approaches. In addition, the static analysis for the malware variants detection has practical time consumption, only tens of second to investigate a thousand of targets. And the time consumption has linear increasing manner to the increase of the number of targets. In conclusion, the proposed system has enough investigation performance for responding the rapidly growing numbers of Android applications. Therefore, it is helpful to protect users from information leakage and economic damages by malware on their smartphone.

Acknowledgment. This research was supported by the public welfare & safety research program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2012M3A2A1051118 2012051118) and the KCC(Korea Communications Commission), Korea, under the R&D program supervised by the KCA(Korea Communications Agency)(KCA-2012-12-911-01-111).

References

1. F-Secure: Mobile threat report q2 2012. Report, F-Secure (2012)
2. Kwon, J., Lee, H.: Bingraph: Discovering mutant malware using hierarchical semantic signatures. In: Proc. of 7th International Conference on Malicious and Unwanted Software, MALWARE 2012 (2012)
3. Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: Proc. of the 2012 IEEE Symposium on Security and Privacy, pp. 95–109 (2012)

4. Microsoft: Atl collection classes (2010), http://msdn.microsoft.com/en-us/library/vstudio/15e672bd(v=vs.100).aspx
5. Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J., Kiraz, O., Yuksel, K., Camtepe, S., Albayrak, S.: Static analysis of executables for collaborative malware detection on android. In: Proc. of the IEEE International Conference on Communications (ICC 2009), pp. 1–5 (June 2009)
6. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. In: Proc. of International Conference on Computational Intelligence and Security (CIS 2010), pp. 329–333 (December 2010)
7. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM 2011), pp. 15–26. ACM (2011)
8. Lee, J., Jeong, K., Lee, H.: Detecting metamorphic malwares using code graphs. In: Proc. of the 2010 ACM Symposium on Applied Computing (SAC 2010), pp. 1970–1977. ACM (2010)
9. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI 2010), pp. 1–6. USENIX Association (2010)
10. Gilbert, P., Chun, B.G., Cox, L.P., Jung, J.: Vision: automated security validation of mobile apps at app markets. In: Proc. of the Second International Workshop on Mobile Cloud Computing and Services (MCS 2011), pp. 21–26. ACM (2011)
11. Blasing, T., Batyuk, L., Schmidt, A.D., Camtepe, S., Albayrak, S.: An android application sandbox system for suspicious software detection. In: Proc. of the 5th International Conference on Malicious and Unwanted Software (MALWARE 2010), pp. 55–62 (October 2010)
12. Barrera, D., Kayacik, H.G., van Oorschot, P.C., Somayaji, A.: A methodology for empirical analysis of permission-based security models and its application to android. In: Proc. of the 17th ACM Conference on Computer and Communications Security (CCS 2010), pp. 73–84. ACM (2010)
13. Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., Glezer, C.: Google android: A comprehensive security assessment. IEEE Security and Privacy 8(2), 35–44 (2010)
14. Enck, W.: Defending users against smartphone apps: Techniques and future directions. In: Jajodia, S., Mazumdar, C. (eds.) ICISS 2011. LNCS, vol. 7093, pp. 49–70. Springer, Heidelberg (2011)
15. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. In: Proc. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM 2011), pp. 3–14. ACM (2011)
16. Whitney, L.: Android's popularity makes it open target for malware, says study. Technical report, CNET (December 2011)