Android Malware Analysis Approach Based on Control Flow Graphs and Machine Learning Algorithms

Mehmet Ali ATICI
Dept. of Comp. Eng. Fac. of Eng.
Gazi University
Ankara, Turkey
mehmetaliatici34@gmail.com

Seref SAGIROGLU
Dept. of Comp. Eng. Fac. of Eng.
Gazi University
Ankara, Turkey
ss@gazi.edu.tr

Ibrahim Alper DOGRU
Dept. of Comp. Eng. Fac. of Tech.
Gazi University
Ankara, Turkey
iadogru@gazi.edu.tr

***Abstract*- Smart devices from smartphones to wearable computers today have been used in many purposes. These devices run various mobile operating systems like Android, iOS, Symbian, Windows Mobile, etc. Since the mobile devices are widely used and contain personal information, they are subject to security attacks by mobile malware applications. In this work we propose a new approach based on control flow graphs and machine learning algorithms for static Android malware analysis. Experimental results have shown that the proposed approach achieves a high classification accuracy of 96.26% in general and high detection rate of 99.15% for DroidKungfu malware families which are very harmful and difficult to detect because of encrypting the root exploits, by reducing data dimension significantly for real time analysis.**

Keywords-Android; mobile security; malware; static analysis; control flow graphs, machine learning

I. INTRODUCTION

Mobile devices contain mobile operating systems like Android, iOS, Symbian, Windows Mobile etc. Among these operating systems, Android is the most widely used one[1]. Detection of mobile malware applications is one of the hot topics in cyber security. According to mobile threat report[2], 277 new threat families have been detected in 2014, 275 of these for Android and remaining ones for iOS and Symbian. Malware detection is made by either static or dynamic analysis in mobile devices as seen from related works in Section 2. In static analysis, source code of the application is examined to detect suspicious code pieces. On the other hand in dynamic analysis behaviors of the applications during run time are also monitored to analyze suspicious behaviors.

Android malware detection in effective way is still a challenging problem. DroidChameleon [1] framework which was developed to test Android anti-malware tools indicates that popular anti-malware tools are vulnerable to even general obfuscation methods and 90% of the signature based approaches don't conduct static code analysis. To improve the success of mobile malware detection it is suggested that detection should be made in semantic manner like analyzing control or data flows.

In this work, a novel approach based on control flow graphs and machine learning algorithms for malware analysis of Android malware families is proposed. The proposed model utilizes grammar representations of control flow graphs of Android applications as input vector.

The contributions of this work are:

- It achieves very high classification accuracy of 96.26% for multi-class classification by static source code analysis.
- It attains very high detection rate of 99.15% for DroidKungfu malware families which are difficult to detect because of encrypting the root exploits.
- To achieve the task, data dimension is consequently reduced by utilizing only about 30 different code chunks.

The paper is organized as follow: In Section 2, related works are introduced. In Section 3, the proposed approach is expressed and implementation details are provided. In Section 4, the results of the experiments are introduced and discussed. Finally, our findings are summarized and future work is given in Section 5.

II. RELATED WORKS

Malware detection especially in mobile devices is one of the hot topics in cyber security. There are a number of works for mobile malware detection that either makes dynamic analysis or static analysis.

MODELZ approach [2] analyses the power consumption behavior of the mobile devices dynamically and produces signatures to detect malware depleting the battery. Andromaly [3] analyses the behaviors of Android malware, collects feature values for a definite time by monitoring the application, and applies different machine learning approaches for classification. MADAM [4] monitors the

[1] http://www.gartner.com/newsroom/id/2875017
[2] https://www.f-secure.com/en/web/labs_global/whitepapers

ISBN: 978-1-4673-9865-7/16/$31.00 ©2016 IEEE

application behaviors in a layered manner and makes real time detection of Android malwares according to 13 features identified from application behaviors. Lu et al. introduced a dynamic Android malware detection approach based on Naïve Bayes (NB) classifier improved by Chi-Square method [5]. The model monitors the predefined malicious behaviors like sending messages or downloading applications in unauthorized manner, composes the feature vector and implements a binary classification of malware samples against benign applications.

In static analysis, Dendroid [6] classified Android malware into corresponding families according to grammars of control flow graph of code chunks and utilized Vector Space Model as in the text mining approaches. It achieved classification accuracy above 94% by implementing 1-Nearest Neighbor (1-NN) algorithm. Zhou and Jiang analyzed the characterization of Android malware applications with respect to issues like repacking, malicious payload etc. [7]. Shen et al. presented a novel signature for Android malware detection based on API calls and system events [8]. Kabakus et al. provided an Android malware detection tool, APK Auditor, based on permissions demanded by applications [9]. It calculates a malware score for each analyzed application through the permissions it demands and calculates a malware threshold value using logistic regression by using stored information on signature database.

Machine learning algorithms have been successfully applied for discriminating Android malware samples from benign applications. Liang and Du proposed an Android Malware Detection approach [10] that uses different permission combinations to classify the benign and malware applications. Liu and Liu proposed a two-layered model based Decision Trees (DT) for Android malware detection [11]. The approach uses permission pairs requested and used successively to decide whether an application is malign or benign. Sheen et al. presented an Android malware detection model that ensembles different classification algorithms like NB, DT, Support Vector Machines (SVM), etc. and applied different feature selection techniques like Chi-Square, IG and Relief [12]. The model implements binary classification of malware and benign Android applications using API calls and permissions. The literature works are summarized in terms of analysis type, algorithms, features, classification type, evaluation metrics and experimental results and are given in Table 1.

TABLE I. SUMMARY OF LITERATURE WORKS

Ref.	Analysis Type	Features	Dataset	Results (in %)
DENDROID	**Static - Multi-class**	**Code Chunk Grammars**	**Android Genome Project**	**94.26 for accuracy**
MODELZ	Dynamic – Binary	Charge flow meter	--	99 for True Positive Rate (TPR)
Andromaly	Dynamic – Binary	Various system metrics	44 Applications	Up to 99 for accuracy
MADAM	Dynamic – Binary	System calls, user idleness	Applications from Google Play	93 for Detection Rate
[13]	Static – Binary	Permissions, code-based properties	Android Genome Project and 1000 benign applications	Up to 93 for accuracy
[5]	Dynamic – Binary	Malicious behaviors	477 Applications	89 for accuracy
[10]	Static – Binary	Permission combination	Android Genome Project and 741 benign application from Google Play	Up to 88 for benign and up to 96 for malware
[14]	Static – Binary	API related features and permissions	McAfee internal repository	About 97 for accuracy
[15]	Static – Binary	API calls, permissions and Linux commands	Android Genome Project and 1000 benign applications	Up to 92.1 for accuracy
[11]	Static – Binary	Requested and used permission pairs	AppChina, Android Genome Project etc.	98.6 for accuracy
[16]	Static - Binary	Permissions	Android Genome Project, Offical Android Market	About 90 for accuracy
[17]	Static - Binary	Permissions	Contagio Mobile and Anzhi Market	Between 75.78 and 85.11
[12]	Static - Binary	API calls and permissions	Android Genome Project	98.8 for F-measure
[8]	**Static - Multi-class**	**Android-specific components**	**Android Genome Project**	**86.36 for Detection Rate**
[9]	Static – Binary	Permissions	Android Genome Project, Drebin	88 for accuracy and 92.5 for specificity.

The proposed approach	**Static - Multi-class**	**Code Chunk Grammar**	**Android Genome Project**	**96.26 for classification accuracy.**

As seen from Table 1, most of the works apply binary classification of Android applications. Binary classifiers may not detect the characteristics of complicated malwares that contain various harmful software [17]. In addition, detection of malware types may provide evidence about possible damages caused by the malware when digital forensic issues applied. There are a few works that apply multi-class detection of Android Malware Families that an application belongs to and the proposed approach in this paper, achieves higher classification accuracy among this kind of approaches.

III. PROPOSED APPROACH

In this work, a novel approach based on control flow graphs and machine learning algorithms for malware analysis of Android malware families is proposed. The proposed approach has utilized the data modeling techniques of Dendroid [6] but maintained simply a Boolean input vector and applied machine learning algorithms for analysis instead of using Vector Space Model as in the text mining approaches.

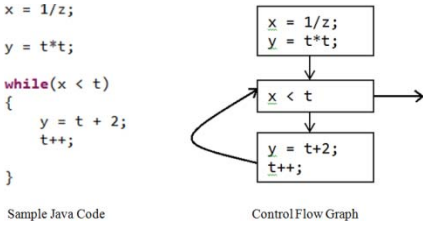

Fig.1. Sample code piece and CFG

In this approach, static malware analysis for Android applications is conducted and control flow graphs of Dalvik bytecode instructions of Android applications have been utilized. Control Flow Graphs (CFG) [18] are directed graphs and can be represented as **G= (B, E)** where **B** indicates basic blocks i.e. the nodes in the graph and **E** indicates execution paths i.e. the edges. Basic blocks are sequences of instructions with just single entry and single exit. An example java code piece and its corresponding control flow graph are given in Fig. 1.

Code chunks are code structures which correspond to single method of a Java class and are firstly used in [6] that utilized Androguard[3] tool to produce CFG representation of the code chunks from Dalvik instructions. Androguard uses the modified version of the grammar introduced in [19]. The grammar used by Androguard and some examples of resultant strings representing code chunks are given in Fig.2 and Fig.3.

Using grammar string representations of Java methods as seen in Fig.3 increases the robustness against obfuscation [6].

[3] https://code.google.com/p/androguard/

```
Procedure ::= StatementList
StatementList ::= Statement | Statement StatementList
Statement ::= BasicBlock | Return | Goto | If | Field | Package | String | Exception
Return ::= 'R'
Goto ::= 'G'
If ::= 'I'
BasicBlock ::= 'B'
Field ::= 'F'0 | 'F'1
Package ::= 'P' PackageNew | 'P' PackageCall
PackageNew ::= 'C'
PackageCall ::= 'M'
PackageName ::= Epsilon | Id
String ::= 'S' Number | 'S' Id
Exception ::= Id
Number ::= \d+
Id ::= [a-zA-Z]\w+
```

Fig.2. Grammar used by Androguard [19]

```
B[]B[F0]B[F0]B[P1P1P1I]B[]B[R]B[P1G]B[P1G]B[P0P1P1P1P1P1P1G]

B[]B[]B[I]B[R]B[]B[P1G]B[]

B[]

B[]B[F0F0I]B[F0F0I]B[P0P1F0P1F0P1F0P1P1I]B[F0P1]B[F0P1]B[R]B[P1F0F0F1G]B[SP1P1G]
```

Fig.3. Sample CFG grammar strings

The proposed approach was implemented on malware samples of Android Malware Genome Project [7] that includes Android malware samples that are categorized into 49 malware families. Table 1 indicates that Android Malware Genome Project dataset has been widely used in malware analysis and detection systems in literature. In this work 33 of these android malware families that contain more than one application and the corresponding 1231 malware samples that belong to these families have been used in the experiments. The malware families containing only one application are excluded because these are not feasible for 2-fold cross validation in training and test. Each malware application consists of code chunks corresponding to Java methods as mentioned before. 1231 malware samples that consist of 84854 distinct code chunks totally are used in this work. CFG representations of these 84854 distinct code chunks have been utilized in the proposed approach to identify the input vector of the neural network. Pseudo code for the proposed approach is given in Fig.4.

```
Android Malware Analyzer
For Each malware
     Create the CFG grammar string representations of methods
     Extract the unique strings and load into database
End For
For Each malware
     Initialize the Boolean vector with 0s and family class label
     For Each unique CFG grammar string
          If malware contains CFG grammar string
               Then Set the Boolean vector element to 1
          End If
     End For
     Append the Boolean vector to data file.
End For
For Each classification algorithms
     Train and test the algorithm on data file
End For
```

Fig.4. Pseudo code of the proposed approach

Once the Android malware is represented as a set of grammar strings denoting control flow graphs of code

chunks i.e. the java methods inside the malware, the input vector of the proposed approach for each malware application **A** is a Boolean vector consisting of 0 and 1 values for each distinct 84854 code chunks where 1 indicates the existence of the code chunk in the malware.

General framework of the proposed approach consists of three basic components including modeling, pre-processing and classification algorithm. In modeling, Android APK files i.e. the applications are decompressed and decompiled into Dalvik instructions. Consequently control flow graph of each application is drawn in which basic blocks represent code chunks that correspond to the Java methods. Once the code chunks are identified, the grammar representations of each code chunks are produced. Thus, each Java method in an Android application is transformed into a string. Since the outputs of the modeling process have already been published[4], these outputs have been used in this work directly without processing the modeling steps.

In pre-processing, the strings representing methods of Android applications are stored in the database with necessary Meta data like application and malware family names. Then existence of each distinct 84854 code chunks is checked for each malware to produce Boolean input vector. Finally, datasets are created for experimental works.

As for the classification, machine learning algorithms including CART (Classification and Regression Tree), PNN (Probabilistic Neural Network), NB and 1-NN were used. CART algorithm attained very high classification accuracy of 96.26%

CART algorithm [20] uses Gini Index calculation for splitting a tree node. For a given dataset D, Gini index is calculated as given in Eq. (1).

$$Gini(D) = 1 - \sum_{i=1}^{c} {p_i}^2 \quad (1)$$

where **c** is number of distinct class labels in dataset **D** and $\boldsymbol{p_i}$ is the ratio of the samples having the corresponding class label. The attribute having the best Gini index value is used as first node for splitting. For all distinct values of an attribute, if all records have the same class label then a leaf node is created and tree construction is finalized for this value. On the other hand a node is created and tree construction continues recursively with respect to the corresponding subset of the dataset **D**. Once a decision tree is constructed, some attributes in the dataset may not be included in the tree meaning that these attributes have no contribution for the classification for this dataset.

PNNs [21] consist of four units; input, pattern, summation and output . Neurons of input and pattern units are fully connected where each connection has a weight. The net input of a single neuron of pattern units is calculated as given in Eq. (2).

$$Z = \sum_{i=1}^{p} x_i . w_i \quad (2)$$

where p is total number of inputs. The output of a neuron of pattern unit is equal to result of a nonlinear activation function with respect to Z. The output of a neuron of pattern unit is equal to result of a nonlinear activation function with respect to net input. Summation unit, as the name implies, sum up the outputs of pattern units for each class. Output unit neuron makes the decision for classification. Learning of PNN is completed after each pattern has been passed once.

NB algorithm is a probabilistic classifier which implements the Bayesian theory by assuming all variables are independent and 1-NN algorithm is an implementation of KNN classifier where K is equal to 1 and detects the class label of the sample according to the training instance which has the minimum distance to that sample [22].

IV. Experimental Results and Discussion

To measure the performance of the proposed approach, an experiment has been conducted on Android Malware Genome Project [7] that includes Android malware samples that are categorized into 49 malware families. Test results were evaluated according to classification accuracy which is ratio of number of correctly predicted samples to number of all test samples. In addition, detection rate i.e. recall or hit rate for each family which is calculated as in context of binary classification as the ratio of number of correctly predicted samples for this family to number of all samples [23] in that family is given in results. In this work 33 of these android malware families containing more than one application and the corresponding 1231 malware samples of which CFG grammar representations could be produced, have been used in the experiments. The reason for excluding the families containing only one malware sample is that if the sample is used in training then there is no existing sample for test and vice versa. Each malware application consists of code chunks corresponding to Java methods as mentioned before. 1231 malware samples used in this work consist of 84854 distinct code chunks totally.

For the experiment, Android malware are represented as a set of grammar strings denoting control flow graphs of code chunks i.e. the java methods inside the malware. The input vector of the proposed approach for each malware application **A** is a Boolean vector consisting of 0 and 1 values for each distinct 84854 code chunks where 1 indicates the existence of the code chunk in the malware.

Therefore the full dataset consists of 1231 records corresponding to 1231 malware samples each having 84855 attributes with class label.

In the experiment, since some malware families contain only two applications, 2-fold cross validation was applied and the proposed approach attained very high classification accuracy of 96.26% by CART algorithm. The classification accuracy values for each algorithm applied in experiment are given in Table II.

[4] http://www.seg.inf.uc3m.es/~guillermo-suarez-tangil/dendroid/codechunks.zip

TABLE II. CLASSIFICATION ACCURACIES

Algorithm	Accuracy (%)
CART	**96.26**
1-NN	90.33
PNN	89.60
NB	89.44

The CART algorithm used about only 30 code chunks of total 84854 code chunks for making a decision. Thus the dimension of the data to be processed and the computational time were reduced significantly.

The classification accuracy values attained by CART algorithm for each malware family are also given in Table III.

TABLE III. DETECTION RATES

Malware Family	Classification hits	Detection Rate (%)
Tapsnake	(1/2)	50,00
BaseBridge	(121/122)	99,18
DroidKungFu3	(309/309)	**100,00**
RogueSPPush	(8/9)	98,88
DroidKungFuSapp	(1/3)	**33,33**
BeanBot	(5/8)	62,50
DroidDreamLight	(46/46)	100,00
KMin	(52/52)	100,00
zHash	(10/11)	90,91
CruseWin	(0/2)	0,00
Pjapps	(44/45)	97,78
HippoSMS	(0/4)	0,00
jSMSHider	(16/16)	100,00
GingerMaster	(1/4)	25,00
GPSSMSSpy	(3/6)	50,00
Plankton	(10/11)	90,91
FakePlayer	(3/6)	50,00
NickySpy	(0/2)	0,00
RogueLemon	(0/2)	0,00
Bgserv	(8/9)	88,89
Asroot	(3/8)	37,50
SndApps	(10/10)	100,00
DroidKungFu1	(33/34)	**97,05**
Zsone	(12/12)	100,00
YZHC	(21/22)	95,45
GoldDream	(47/47)	100,00
DroidKungFu2	(29/30)	**96,66**
ADRD	(21/22)	95,45
DroidKungFu4	(96/96)	**100,00**
AnserverBot	(186/187)	98,93
DroidDream	(15/16)	93,75
Gone60	(5/9)	55,55
Geinimi	(69/69)	100,00

As seen from Table III, the proposed model has also achieved very high detection rate of 99.15% for DroidKungfu malware families which are difficult to detect because of encrypting the root exploits [7]. DroidKungfu utilizes obfuscation techniques definitely [6] and the attained high detection rate of 99.15% is probably consequence of improved robustness against obfuscation provided by utilizing code chunk representations. In addition to this, Table IV indicates that the proposed approach also attains better results in other malware families with respect to low misclassification hits when compared the results of detector proposed in [8] that use the same dataset as this work.

TABLE IV. MISCLASSIFICATION HITS

Malware Family	The proposed approach	Work in [8]
DroidDreamLight	**(0/46)**	**(0/46)**
HippoSMS	(4/4)	**(0/4)**
Plankton	**(1/11)**	**(1/11)**
Bgserv	(1/9)	**(0/9)**
Zsone	**(0/12)**	(2/12)
YZHC	**(1/22)**	(3/22)
GoldDream	**(0/47)**	(8/47)
DroidKungFu2	**(1/30)**	(5/30)
Geinimi	**(0/69)**	(15/69)

For some malware families, detection rates are too low but this is expected because number of samples belong to these families may not be sufficient for learning of the algorithms. In Table IV, we didn't give the results of the works that use different datasets since the comparison was not possible.

V. CONCLUSIONS AND FUTURE WORKS

In this work, a novel approach based on control flow graphs and machine learning algorithms for malware analysis of Android malware families has been successfully introduced. It utilizes grammar representations of control flow graphs of Android malware as input vector.

Experimental results have shown that the proposed approach achieves higher classification accuracy and uses only about 30 code chunks as input vector which consequently reduces the dimension of the data to be processed and decreases the computational time. In addition, the proposed approach obtains more successful detection rates for some challenging malware families like DroidKungfu. As for the future work, by using the advantage of representing the java methods of malware samples as strings, we will utilize string similarity algorithms to handle variants of the methods reused in malware applications for Android malware analysis.

REFERENCES

[1] V. Rastogi, Y. Chen, X. Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks", IEEE Transactions on Information Forensics and Security, vol. 9, pp. 99–108, 2014.

[2] H. Kim, K.G. Shin, P. Pillai, "MODELZ: Monitoring, Detection, and Analysis of Energy-Greedy Anomalies in Mobile Handsets", IEEE Transactions on Mobile Computing, vol. 10, pp. 968–981, 2011.

[3] A. Shabtai, U. Kanonov, Y. Elovici, Y., C. Glezer, Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices", Journal of Intelligent Information Systems, vol. 38, pp. 161–190, 2012.

[4] G. Dini, F. Martinelli, A. Saracino, D. Sgandurra, "MADAM: A Multi-level Anomaly Detector for Android Malware", Computer Network Security, Lecture Notes in Computer Science., vol.7531, pp. 240–253, 2012.

[5] Y. Lu, P. Zulie, L. Jingju, " Android malware detection technology based on improved Bayesian Classification", Third International Conference on Instrumentation, Measurement, Computer, Communication and Control, Shenyang, pp. 1338–1341, 2013.

[6] G. Suarez-Tangil, J.E. Tapiador, P. Peris-Lopez, J. Blasco, "DENDROID: A text mining approach to analyzing and classifying code structures in Android malware families", Expert Systems with Applications, vol. 41, pp. 1104–1117, 2014.

[7] Y. Zhou, X. Jiang, "Dissecting Android Malware: Characterization and Evolution", 33rd IEEE Symposium on Security and Privacy (Oakland 2012). IEEE, San Francisco, CA, USA, pp. 95–109, 2012.

[8] T. Shen, Y. Zhongyang, Z. Xin, B. Mao, H. Huang, "Detect Android Malware Variants using Component Based Topology Graph", IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, China, pp. 406–413, 2014.

[9] A.T. Kabakus, I.A. Dogru, C. Aydın, "APK Auditor: Permission-based Android malware detection system", Digital Investigation, vol. 13, pp. 1-14, 2015.

[10] S. Liang, X. Du, "Permission-combination-based scheme for Android mobile malware detection", IEEE International Conference on Communications (ICC), Sydney, Australia, pp. 2301–2306, 2014.

[11] X. Liu, J. Liu, "A Two-layered Permission-based Android Malware Detection Scheme", 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, Oxford, pp. 142–148, 2014.

[12] S. Sheen, R. Anitha, V., Natarajan, "Android based malware detection using a multifeature collaborative decision fusion approach", Neurocomputing, vol. 151, pp. 905–912, 2015.

[13] S.Y. Yerima, S. Sezer, G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection", IET Information Security, vol. 8, pp. 25–36, 2014.

[14] S.Y. Yerima, S. Sezer, I. Muttik, "Android Malware Detection Using Parallel Machine Learning Classifiers", Eighth International Conference on Next Generation Mobile Applications, Services and Technologies, Sydney, pp. 37–42, 2014.

[15] S.Y. Yerima, S. Sezer, I. Muttik, I., "A New Android Malware Detection Approach Using Bayesian Classification", IEEE 27th International Conference on Advanced Information Networking and Applications, Barcelona, pp. 121–128, 2013

[16] Z. Xiaoyan, F. Juan, W. Xiujuan, "Android malware detection based on permissions", International Conference on Information and Communications Technologies (ICT 2014), Nanjing, pp. 1–5, 2014.

[17] W. Liu, "Mutiple classifier system based android malware detection", International Conference on Machine Learning and Cybernetics (ICMLC), Tianjin, pp. 57–62, 2013.

[18] F.E. Allen, "Control flow analysis", ACM SIGPLAN Notices, vol. 5, pp. 1–19, 1970.

[19] S. Cesare, Y. Xiang, "Classification of malware using structured control flow", Proceedings of the eighth Australasian symposium on parallel and distributed computing, Brisbane, Australia, pp. 61–70, 2010.

[20] L. Rutkowski, M. Jaworski, L. Pietruczuk, et al., "The CART decision tree for mining data streams", Information Sciences, vol. 266, pp. 1–15, 2014.

[21] D.F. Specht, "Probabilistic Neural Networks and the Polynomial Adaline as Complementary Techniques for Classification", IEEE Transactions on Neural Networks, vol. 1, pp. 111–121, 1990.

[22] M.J. Islam, Q.M.J. Wu, M. Ahmadi, M.A. Sid-Ahmed, "Investigating the Performance of Naive- Bayes Classifiers and K-Nearest Neighbor Classifiers", International Conference on Convergence Information Technology, Gyeongju, pp. 1541–1546, 2007.

[23] T. Fawcett, "An introduction to ROC analysis", Pattern Recognit. Lett., vol. 27, pp. 861–874, 2006.