International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

MDROID: ANDROID BASED MALWARE DETECTION USING MCM CLASSIFIER

Sushma Verma
SAG
DRDO, New Delhi, India

Sunil Kumar Muttoo
Department of Computer Science
FMS, University of Delhi, Delhi, India

S.K. Pal
SAG
DRDO, New Delhi, India

***Abstract*— Malware analysis and detection has become a prime research area in the case of smartphones, particularly based on android due to its widespread usage and increase in the number of malwares involving huge monetary gains. The exploding number of Android malware calls for automated analysis of the systems. There are two common techniques used for detecting malware, signature based and behaviour based. Signature based detection uses a sequence of bytes that appear in the binary code to identify and detect a family of malware. Behaviour based detection uses features/ artifacts created by malware during execution for identification. In this paper, we propose a new malware classification method based on semantic similarity between two common subgraphs which is effective for the detection and analysis of new threats for which signatures are not available, A behaviour graph is obtained by capturing suspicious API calls during the execution (in a sandboxed environment). We use a labelling mechanism for the API calls which will be regarded as a signature for malicious activity. Selected features are used to train an MCM classifier. On several benchmark datasets, the MCM classifier yields detection accuracy of 97% even with using one-tenth the number of support vectors used by SVMs.**

Keywords— ***Android Malware Analysis, API calls, Feature space embedding, Graph kernel, MCM classifier***

I. INTRODUCTION

Android is becoming a target to a growing number of attacks and malicious applications due to its popularity.[40]. The goal of the attackers is to steal private information, transferring credit into their ac-count by subscribing to premium services, unwarranted premium-rate subscription of SMS services and advanced frauds. Most of the current commercial antivirus for malware detection are based on Static analysis which fail to detect zero-day malwares.[42] Signature based technique treats malwares as sequences of bytes that performs well for known malwares. Generally malware uses obfuscation as well as packing techniques to make static analysis harder[8]. Static analysis based detection methods can be easily by passed by simple code obfuscation because it ignores programs functionality such as APIs or function calls.[7] The obfuscation used by Android applications hides system activities by calling functions in native libraries written in C / C++ which is outside the Dalvik /Java runtime library. In dynamic analysis the source code is executed in a controlled environment, often called sandbox.[10] Dynamic analysis can counter obfuscation techniques but can be bypassed by runtime detection methods. Function-call graph was created from the disassembled code of program, in which all vertices represent functions in the program. Each function is a set of API calls . Edges represent relationship among functions. The instruction sequences of the malware binary depicting the structure and the functions are converted into a function-call graph. Function-call graph[38] abstracts away byte or instruction level details which act as an signature for the malware. It can be used to classify the malware variants. We propose a technique which can identify the semantic similarity between two malware programs through the use of function-call graph. We have used a labelling mechanism for the API calls in the function call graph which will be regarded as a signature for malicious activity. A Minimal complexity machine is then used as a classifier to distinguish between benign and malicious applications. Experimental results show that the performance of this classifier is better than the classical SVM in terms of generalization accuracies on a number of selected features and number of support vectors required. In an experimental result on a total of 1200 malware samples, this classification approach is more effective for a resource constrained device such as smartphones, allowing a detection of 97 % of the malware families with only 1% false positives A sketch of our contributions are mentioned below:

– **Feature Space Extraction:** Features are extracted by reverse engineering the Android application package file (APK)[14]. We decompile classes.dex file through dex2jar and then use jd-gui tool to analyse java source codes of jar/class file. The components of APK file comprise of AndroidMan-ifest.xml, Classes.dex, res directory, lib directory, META-INF directory, and resources. We extract the feature sets in the form of permissions and various API calls which help us in detecting the malicious activities using the Androidmanifest.xml.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

– **Extracting Call Graph Function And Labelling:** The function call graph for a malware apk is extracted. Each function call of the application is represented by a node .Nodes are labeled according to the API calls contained in their corresponding functions. A hash-value is calculated over the node and direct neighboring nodes for each node in the function call graph which is treated as a label for that node.[3] This neighbourhood hash function also explicitly enumerates the occurrences of graph substructures.

– **Classification Of Malware Using MCM For Accurate Generalization:** A classification module is developed with the aim of accurately declaring the app as a benign or malicious one. It achieves this by using a set of known permissions and API calls which are embedded in a feature space with the goal of finding a representation that is equivalent to a graph kernel. MCM classifier algorithm[27] is then used to classify an application into malicious and benign one. Though it is difficult to transform graphs to feature vectors without loss of structutal information contained in the graph, MCM uses a graph kernel based approach to graph learning while avoiding the explicit representation of the graph in high dimensional feature space[31]. It is based on low VC dimension which leads to good generalization.

Rest of the paper is organized as follows: Section II: Details the related research work. Section III: Introduces our proposed technique of detecting Android malware by analyzing function call graphs and labelling them for forming signatures. Section IV presents the training and analysis. Section V details the experiments and the test results followed by detection performance in section VI and finally the conclusion in section VII.

II. RELATED RESEARCH WORK

There has been a continuous increase in malware based attacks in the past decade leading to a significant increase in research on malware detection for smartphones. SCANDAL[12] proposed by Kim described a Static Analyzer for detecting privacy leaks in Android applications based on the optimized method of advanced permission based detection. RiskRanker [13] detects zero day malwares by analysing untrusted apps in the Android market according to potential security risks in a two order risk analysis framework. DroidAPIMiner[20] and Drebin[2] classify applications based on static analysis of features learned from various benign and malicious applications. In [37], a technique based on machine-learning algorithms was proposed by zarni for detection of malicious Android applications in Android. They suggested that the best representation of executables is the combination of both permissions and features from the Manifest file. Several permission features are extracted from various downloaded applications from Android. The research work focused on dynamic analysis of Android malware includes TaintDroid [18] and DroidScope[15] .TaintDroid by W. Enck focuses on taint analysis and tracks the flow of privacy sensitive data through third-party applications by leveraging Android's virtualized execution environment. DroidScope examines application at different layers of the platform. Both approaches provide detailed information about the behavior of applications but they require too many resources and cannot be deployed on smartphones directly. In pBMDS[22] a behavior-based malware detection sys-tem was proposed that correlates users inputs with system calls related to SMS/MMS sending to detect anomalous activities. DroidDolphin [23] uses a combination of static and dynamic analysis and repack-ages the application by inserting the monitoring code. However the increasing use of emulator detection technology in malware evades the dynamic analysis methods.

Another model DroidAnalytics is a signature based analysis system to automatically collect and analyze android malware [25]. It uses a multi-level signature algorithm to extract the malware feature based on their semantic meaning at the opcode level[34]. It is effective in analyzing repackaged and metamorphic malwares. Apposcopy [4] aims at describing the semantics-based malware detection approach which analyses the program statically and defines a high level language for specifying signatures of the mal-wares. It takes in account for the Inter-Component Call Graph of a malware to analyse its control flow. DroidChameleon[26] evaluated commercial malwares for their resistance to common obfuscation techniques. They demonstrated that most of the android anti-malwares were unable to detect the programs even after small transformation in the program.

In [28], a technique was proposed based on static and semantics aware malware detection that attempts to detect code obfuscation by identifying semantically equivalent instruction sequences in the malware variants. However attacks using the equivalent instruction replacement and reordering are still possible as it requires exact matching between the template and application instructions.

SmartSiren [24] proposed by Cheng is a collaborative virus detection and alert system that uses a statistical analysis on the collected data to detect abnormal communication patterns such as excessive daily us-age of SMS/MMS messages. The difficulty of manually creating and updating detection patterns based on static or dymanic analysis for Android has motivated the use of machine learning techniques[16]. Several techniques have been proposed that analyze applications automatically using machine learning methods [11]. In[19], a machine-learning based framework CROWDROID was developed that detects Trojan-like malware on Android smartphones. It analyzes the number of API invocations and system call count that has been used by an application during the execution of an action. However, their detection methodology can be easily detected by malware as it modifies the application under analysis.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

In MAMA[29] manifest analysis for malware detection in android is explained . They evaluated the capacity of these two feature sets viz uses-permissions and uses-features in the manifest file to detect malware using machine-learning techniques. Machine learning algorithms has been considerably used for anomaly detection. [11]. Recently, Support Vector Machines (SVMs),a supervised learning algorithm based on the pioneering work of Vapnik [32] and Joachims [33] on statistical learning theory have been successfully applied in a number of classification problems. The emergence of mobile malware that spread via SMS/MMS messaging and Bluetooth is increasing at an alarming rate thereby requiring novel detection methods. Though Random Forest and Support Vector Machines(SVM) are amongst the most widely used machine learning techniques today for malware detection. The SVM[30] is used widely with several variants such as the maximum margin L1 norm SVM , and the least squares SVM (LSSVM) based on solving the quadratic programming problem. However, taking into account the limitations and resource constraints of smartphones, we present a machine learning based classifier system for the detec-tion of malware on Android devices based on minimal complexity machine. We integrate the call graph labelling approach with a new lightweight classifier for smartphones that accounts for unknown malware behaviors. The goal of our work is to develop a malware detection framework for android that overcomes the limitations of signature-based detection while addressing resource constraints of smartphones.

III. PROPOSED METHODOLOGY

This section describes the overall methodology.

1. **Call graph extraction :** The function call graph for an application will be extracted, which contains information about the nodes and edges present. Each node will be labelled according to the instruction they contain and function they represent. It employs 2 steps:

Step 1 Unpack the malware and disassemble using dex2jar. Extract API calls and permissions used for the apk using jd-gui as depicted in fig 1.

Step 2 Extract the function call graph of an android application with the details of edges and nodes using Gephi as shown in fig 2.

2. **Hashing of neighbourhoods and Labelling**: For every node in a graph, there will be a set of edges. A hash value is generated for each node based upon the labels of nodes themselves as well as their neighbours. This makes it easy to include not just the properties of the node but also the occurrences of substructures traversed

3. **Count sensitive Graph Kernel**. Neighbourhood hash values for unrelated nodes can be same leading to accidental hash collision and resulting in positive semi-definitiveness of the kernel. In order to re-solve the problem of hash collision, we use graph kernel based on count of common substructures.

4. **Feature space embedding**. Features are embedded using an explicit map inspired by the count sensitive neighborhood hash graph kernel introduced by Hugo Gascon [3] Employ an explicit mapping inspired by a linear-time graph kernel to efficiently map call graphs to an explicit feature space. The map is designed such that evaluating an inner product in the feature space is equivalent to computing the respective graph kernel.

5. **Learning and feature analysis**. A multiple complexity machine is then trained to learn a detection model that is able to classify applications as benign or malicious. The classifier which takes malicious features embedded in a graph kernel along with a behaviour signature database detection of known malwares. The model is then deployed in the handsets to detect the malware apks.

In an empirical evaluation on a total of 1200 malware samples, this approach is shown to be highly effective, enabling a detection of 97% of the malware families with only 1% false positives. The methodology is used to detect the presence of an instance of a malware in an android application by considering its function call pattern [35] and ignoring the syntax of a code thus making it resilient to various obfuscation techniques.[8] It aims to find the presence of the malware instance by concentrating on two generally observed facts:

1. Malicious functionality of an Android program is present only on a very small number of functions.

2. Number of times, the function with malicious instance called is much higher than the normal functions. To understand more about every step, let us talk about each of them in detail.

To understand more about every step, let us talk about each of them in detail.

A) Call Graph Extraction and Labelling:

We first begin by disassembling the application with the help of dex2jar and jd-gui [14]. In order to extract the malicious features[1], we analyse Androidmanifest.xml using Android asset packaging tool (AAPT). The extracted static features and dynamic features are represented as strings, which cannot be fed to classifier directly. For example, a malware sending premium SMS messages may contain the requested permissions “SEND_SMS", and the hardware components “Android.hardware.telephony"

Feature sets are extracted from the AndroidManifest file. The information stored in this file can be efficiently retrieved using AAPT to extract the following feature sets:

Feature Set 1: Hardware Components: If an application is requesting access to specific hardware components like camera, touchscreen, GPS module, it indicates some harmful behaviour leading to security breach. An attacker can collect

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

location information and send it over the network using GPS and network access.

Feature Set 2: Permissions: Whenever an application is installed in Android, it requests the permissions which allows an application to access related resources. A malicious application can make use of these permissions to access resources leading to security implications. For example, most of the malwares like DroidKungfu, Fakeplayer, Basebridge and JSMHider sends premium SMS messages using SEND_SMS permission.

Feature Set 3: App components: An android application makes use of four different components:

Activities, services, content providers and broadcast receivers.[20] These components are declared in the manifest. A malware can make use of these components to perform the suspicious activity. For example, Basebridge malware activate three service - AdSmsService, BridgeProvider and PhoneService to com-municate with control server,also block messages to do malicious activity without user‘sends knowledge.

Feature Set 4: Filtered intents: Intents provide interprocess and intraprocess communication allowing information about events to be exchanged between different components and applications. As malware uses these intents for malicious communication, it is listed in the manifest file. A typical example is Gin-Master which root devices to escalate privileges, steal confidential information and send it to a remote website, plus install applications without user interaction.

Feature Set 5: Restricted API calls: Use of restricted API calls for which the required permissions have not been requested reveals malicious behaviour. Example includes the malware using root exploits such as GinMaster to bypass the limitations imposed by the Android platform.

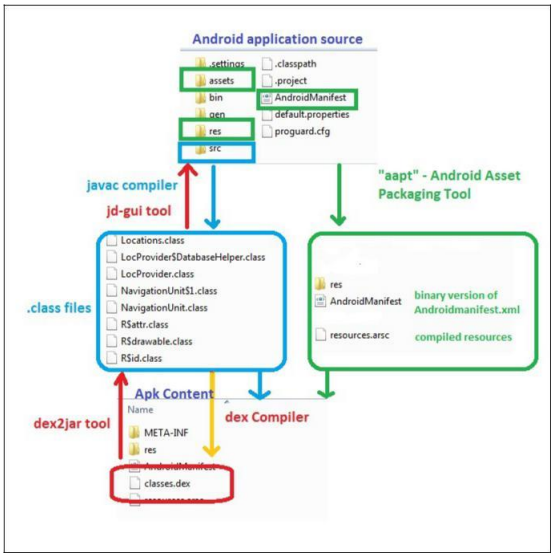

Fig. 1. Disassembling APK

Feature Set 6: Used permissions: We extract the set of API calls used and the corresponding subset of permissions that are both requested and actually used. Stowaway, an automated tool by Felt et. al [9] determines the set of API calls used by an application and maps to permissions.

Feature Set 7: Suspicious API calls: Certain API calls which allow access to sensitive data or resources of the smartphone leading to malicious activity are gathered in a separate feature set. Some of the examples include API calls for the following:

– Accessing sensitive data, such as getDeviceId() and getSubscriberId()

– Accessing sensitive data, such as getDeviceId() and getSubscriberId()

– Network communication, execHttpRequest() and setWifiEnabled()

– Sending and receiving SMS, such as sendTextMessage()

– Execution of external commands like Runtime.exec()

– Obfuscation, like Cipher.getInstance()

Feature Set 8: Network addresses: Network connections are used to retrieve commands or filter data such as all IP addresses, hostnames and URLs collected from the device. Some of these addresses are involved in botnets and serve as a very important feature for the detection of a malicious behaviour.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

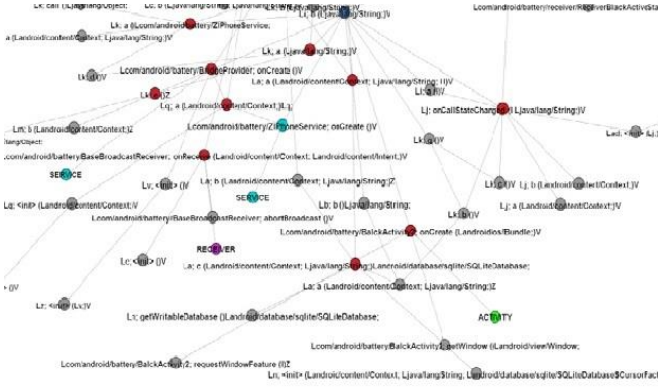

Fig. 2. Function Call Graph for the malware

We then obtain its function call graph with the details of edges and nodes.[21] To optimize our frame-work each node is assigned a label in the form of a short bit sequence of 8 bits. Mathematically, each graph can be formalized as a tuple of four factors namely nodes, edges, label set, and the labelling function. Each graph G can be represented as G = (V,E,L,l) where V is the finite set of nodes, and each node v2 V is associated with one of the applications´ functions; E: V1 ! V2 denotes the set of directed edges, where an edge from a node v1 to a node v2 indicates a call from the function represented by v1 to the function represented by v2. Finally, L is the multiple set of labels in the graph and l: V!L is a labelling function, which assigns a label to each node by considering a set of features from the Feature set which represents the function it performs. An application is mapped to a vector space of malicious features by constructing a vector B(x), such that for each feature extracted from the application.

We define a label function for a node as:

$$L(v) = B(x_1), B(x_2), B(x_3), B(x_4), \ldots\ldots\ldots\ldots B(x_n) \quad (1)$$

n=number of features from the set of Feature Set described above.

$$B(x) = \begin{cases} 1, x \in featureset \\ 0, x \notin featureset \end{cases} \quad (2)$$

Where x_i denotes the presence or absence of that feature set in a node. Thus we formulate a boolean expression that capture the functionality of a malware using the feature sets described above and the dependencies between features. We illustrate this with an example of a malicious application which sends premium SMS messages using permissions and hardware components. A corresponding vector repre-sentation for this application looks like this:

$$\phi(x) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad (3)$$

B) Hashing of Neighbourhoods

A malware performs a list of activities in the course of its lifecycle that may appear to be harmless when analyzed in isolation. So a malware cannot be detected by the activity of a standalone function. Thus we strive to incorporate composition of a function that not only includes the label of node itself but also includes labels of neighbouring nodes. We compute a neighbourhood hash over all of its direct neighbours in the call graph. We use a procedure inspired by the neighbourhood hash graph kernel (NHGK) originally proposed by Hido and Kashima.[36] The NHGK is a so called decomposition kernel as defined by Haussler [39] because it is a kernel operating over a large set of sub graphs. Its complexity is fairly low and expresses the graph in a very readily readable form. Moreover, it is able to run in linear time in the number of nodes.

The main ideology behind the NHGK is to package the information of all the neighbouring nodes and then further incorporating that value with label's original node to form a new hash. The algorithm used to calculate hash for a node is given by

$$h(v) = r(l(v)) \prod \left\{ \prod_{Z \in Vv} l(z) \right\} \quad (1)$$

Thus h(v) will provide us with a new hash or a new label for a node which is a function of G = (V; E; L; h(:)) where h(.) is a hash function which assigns the label L to each node by considering the function associated with it. The neighbourhood hash of order p can then be defined recursively as $G^{p+1} = h(G^p)$. Choosing p larger than one still allows to construct a valid decomposition kernel [3]. Since incorporating path length of more than one, there is a risk of overlapping substructures and thus the values of bits observed might lead to wrong results. Therefore taking values of hash over the path in call graph of length of only 1 is a better option than taking it over iterative sequence of functions in the call graph.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

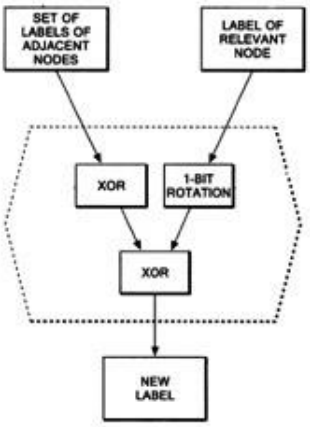

Fig. 3. Algorithm for NHGK

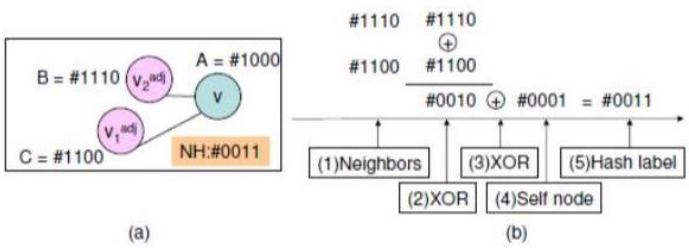

Fig. 4. An example of simple neighborhood hash. (a) A node v having two neighbours. (b) The procedure to compute the neighborhood hash for node v using XOR and ROT

C) Graph Kernel

The previous section covered how to construct a function call graph from a given application and use a labeling function for the vertices to represent a functionality in the subgraphs through the use of neighbourhood hash. The next step is to find the similarity between the graph G(1)representing the malware and graph G(h) and of the application. So the malware detection problem transforms to a graph classification problem. The graph classification problem is linked to the graph similarity problem are computationally very hard. Traditional techniques for finding the graph similarity such as the Graph Edit Distance which measures the number of basic operations needed to transform one graph into another are NP-complete[6]. Kernel-based Support Vector Machines[31] have been proven extremely powerful for classification tasks and Graph Kernels have emerged as a solution to let the SVM operate efficiently in the graph space. Graph kernels measure the similarity between two graphs without an explicit construction of the feature vectors. In order to use the current machine learning techniques used to classify graphs G(h) and G(1), we make use of the graph kernel. The aim is to find the number of common subgraphs related to the features, we need to classify the samples as a function of their shared common substructures. In order to calculate the degree of similarity between the graphs, the NHGK evaluates the count of common identical substructures in two graphs, which after hashing gives the number of shared node labels. Since multiple nodes can have same label or hash, kernel value can be represented as the size of the intersection of the multisets L_h and L_1 for two function call graphs G_h and G_1 which can be defined as a function.

$$K_p(G_h, G_1) = |L_h \cap L_1| \quad (4)$$

To classify the program as malicious and benign, we can calculate the above function to find the similarities amongst different set of graphs.

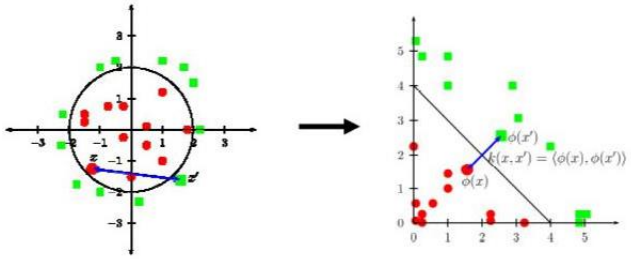

Fig. 5. Graph Kernel

D) Feature Space Embedding

We then input the features in terms of the neighbourhood hash obtained for the nodes to make use of classifier to detect the application as malicious or benign. We embed every kernel K in the feature space whose inner product is equivalent to the graph kernel. The neighborhood hash graph kernel as represented in Fig 5, evaluates the count of common identical substructures in two graphs. Now we create an explicit representation for the kernel represented by histogram H which is fed to the classifier. In order to understand the decision of the classifier malicious and benign, we map the histogram H of labels represented in binary form in a graph to a vector in the following way, where M is the maximum value for all the histograms in the dataset and a is the value of each bin in H. We first sort the labels and formulate a histrogram H for multiset L_h as $H(L_h)$ which is of the form (a_1, a_2, a_3, a_4,…… a_p) where a_i ϵ to the set of natural numbers and denotes the frequency of ith hash in G and p is the number of elements in L_i. Thus the number of shared nodes will be equal the the minimum of a_i amongst the two graphs denoted by the function.

$$S(H_1, H_h) = \sum_i^p min(a_h^i, a_i^h) \quad (5)$$

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 8, ISSN No. 2455-2143, Pages 206-215
Published Online June - July 2016 in IJEAST (http://www.ijeast.com)

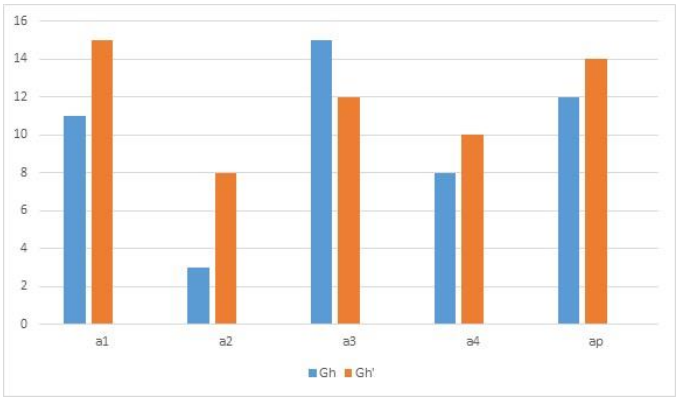

Fig. 6. Histogram for the frequency of the labels a1, a2 to ap in Graph (G_h and G_1)

This provides an easy way to denote the intersection between the two multisets. This is known as multiset intersection. This can be easily observed that function $K_p(G_h; G_1)$ is similar to the function $S(H_1; H_h)$. We further use a histogram function (H) for feature mapping in such a way that S is an inner product in the induced vector space initially proposed by Barla. For the purpose, each histogram is mapped to a P-dimensional vector (H) (Eqn 7). Each bin i of the histogram is associated with M dimensions in the vector space.

$$\emptyset(H) = \left(\underbrace{\overbrace{1,\dots,1}^{a_1}\overbrace{0,\dots,0}^{M-a_1}}_{bin1} \dots, \underbrace{\overbrace{1,\dots,1}^{a_N}\overbrace{0,\dots,0}^{M-a_P}}_{binP} \right)$$

Where M is the maximum value amongst all the histogram bins and P is the number of bins present in the data set. Thus the number of elements in the vector space is PM.

Suppose the frequency of the given label is a_i. These dimensions are marked as 1 for a_i number of terms and 0 for the rest M - a_i terms left. Thus the whole graph is represented in a vector space in the form of 0's and 1's which makes it very easy to analyse and also to recognize the pattern as for each bin, the sum of elements will be its frequency or height of the histogram. Thus the ϕ(H) function represents the similar graph. Comparing the two graph on the basis of the function ϕ(H) will be even more comfortable as shown in Fig 6. Thus the neighbourhood hash graph kernel can be represented as

$$K_p(G_h, G_1) = S(H_h, H_1) = \langle \phi(H_h), \phi(H_1) \rangle$$

One important advantage to represent the program in the form of (H_h; H_1) provides us a way to analyse using MCM. It provides a method to handle programs with thousands of edges and nodes.

IV. TRAINING AND FEATURE ANALYSIS

While Support Vector Machine produces state of the art classi-fier, VC dimension of a SVM can be unbounded. The VC dimension measures the complexity of a learning machine, and a low VC dimension leads to good generalization. So we make use of Minimal Complexity Machine proposed by Jaydeva [27] to learn a hyperplane classifier by minimizing an exact, or bound on its VC dimension. It considers each data point as a vector in space and uses the principles of regressions to categorize them into two. There exists a hyperplane that can classify these points with zero error.[27] To serve our objective, we implement the methods of Graph Kernels, which is a powerful machine learning framework to provide inner-product in a graph, and also find similarities amongst the various graphical structures. Addition of machine learning classifier will add self-efficacy to this system making it, self-reliant. Graph kernel will provide a platform for our classifier to work in a graphical space and MCM can be used to classify the program as malicious and benign on the basis of results obtained.

V. EXPERIMENTS, DATA SETS AND RESULTS

For all experiments, we consider a dataset of real Android applications [5] and real malware acquired from the Google Play Store, contagiominidump[43], virus total and various other sources, such as An-droid websites, malware forums , security blogs and Android Malware Genome Project[44]. We use Virus Total[41] to determine malicious and benign applications. We ensure that the applications are accurately split into benign and malicious samples by flagging the application as malicious even if one of the ten scanners falsely labels a benign application as malicious. We have used 20 malware families in our dataset as listed in the Table 1.

Label	Malware Family	Samples	Detection Accuracy	Label	Malware Family	Samples	Detection Accuracy
A	Anserverbot	30	83.00	N	SVPeng	27	82.00
B	Asroot	42	94.00	O	jSMSHider	10	85.00
C	BaseBridge	55	97.21	P	Android Zeahache	12	89.34
D	Droidkungfu	67	91.02	Q	Plankton	11	88.48
E	DroidDream	48	98.00	R	SMSReplicator	51	96.89
F	Nickspy	32	92.00	S	TapSnake	35	93.68
G	Genimi	63	99.00	T	YZHC	20	84.39
H	GingerMaster	54	95.00	U	Zitmo	62	99.12
I	GoldDream	32	89.12				
J	FakePlayer	52	98.74				
K	HippoSMS	13	78.56				
L	jSMSHider	10	85.12				
M	LoveTrap	23	79.83				

Table 1

List of Malware Samples and their Detection accuracy

VI. DETECTION PERFORMANCE

MDROID is able to reliably detect all families with an average accuracy of 97% and a false-positive rate of 1%. In particular, all families show a detection rate of more than 90%. Its detection rate depends on the number of samples taken for training the detection model. We have trained our model on the samples taken from a set of known malware families listed in the table above. The features extracted from the set of known malware families are examined and the feaures with the highest

contribution to the classification decision are averaged over all members of known families and then used for the detection of the unknown malware. The kernel trick of MCM allows the data instances to be projected into a higher dimensional space and find a hyperplane in that space which is same as a non-linear hyperplane in the original d-dimensional space. Our results show a significant improvement over other existing machine learning methods. The results of the experiments are shown in Fig 7.

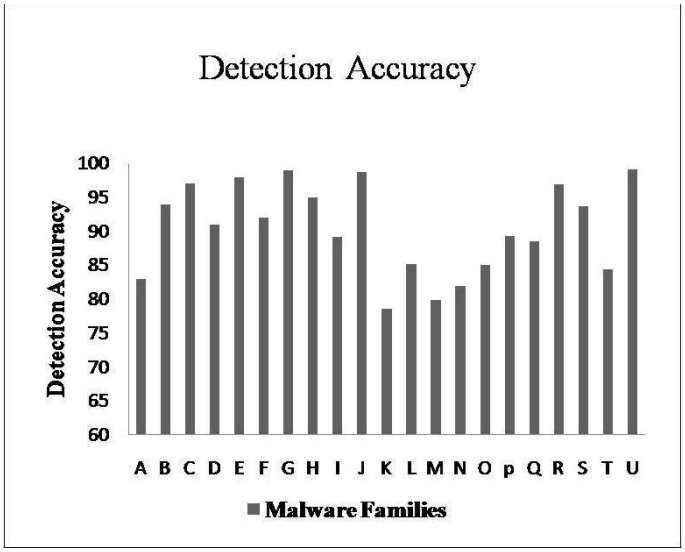

Fig. 7. Detection Accuracy of Malware Families

VII. CONCLUSION

Security attacks on smart phones are become smarter and devastating as more people are switching to smart phones for their serious applications. With an exponential growth in unknown malware, there is a need to establish malware detection methods that are both robust and efficient. As the vast majority of mobile malware targets the Android platform, this work focuses on structural detection of Android malware. However, the method presented can be adapted to other platforms with minor changes in the feature sets extracted. Our method employs static analysis approach for extracting the feature set and an explicit feature map using a labelling function inspired by the neighbourhood hash graph kernel to represent malicious applications based on their function call graphs. We have presented a machine learning approach based on hyper plane classifier MCM, especially suitable for resource constrained devices such as smartphones. We have conducted experiments on various malware datasets and shown that the trained classifier outperforms classical SVM in terms of generalization accuracies on selected datasets. Future work would investigate the classifier performance with larger sample sets as more malware samples are discovered in the wild. Further studies would also investigate performance improvement via prior incorporation of expert knowledge for feature selection.

VIII. REFERENCE

[1] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, Ainuddin Wahid Abdul Wahab, A review on feature selection in mobile malware detection, Digital Investigation: The International Journal of Digital Forensics **&** Incident Response, Volume 13, Issue C, June 2015, pp.22-37.

[2] Daniel Arp, Konrad Rieck, et al.Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket, 21th Annual Network and Distributed System Security Symposium (NDSS), February 2014.

[3] Hugo Gascon , Fabian Yamaguchi , Daniel Arp , Konrad Rieck, Structural detection of android malware using embedded call graphs, Proceedings of the 2013 ACM workshop on Artificial intelligence and security, November 04-04, 2013, Berlin, Germany

[4] Feng, Yu, et al. Apposcopy: Semantics-based detection of android malware through static analysis. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014.

[5] CuckooDroid - http://cuckoo-droid.readthedocs.org/

[6] Kinable, Joris, and Orestis Kostakis. Malware classification based on call graph clustering. Journal in computer virology 7.4 (2011): 233-245.

[7] Moser, A., Kruegel, C., Kirda, E. (2007, December).Limits of static analysis for malware detection. In Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual (pp. 421-430). IEEE.

[8] You, I., Yim, K. (2010, November). Malware obfuscation techniques: A brief survey. In 2010 International conference on broadband, wireless computing, communication and applications (pp. 297-300). IEEE.

[9] P. Felt, E. Chin, S. Hanna, D. Song, and D. Wag- ner. Android permissions demystied. In Proc. of ACM Conference on Computer and Communications Security (CCS), pages 627 638, 2011.

[10] Bose, X. Hu, K. G. Shin, and T. Park, Behavioral detection of malware on mobile handsets. Proc. 6th international Conference on mobile systems, applications and services. Breckenridge, CO, USA: ACM, 2008, pp. 225-238. (EISIC). IEEE; 2012.

[11] Sahs, Justin, and Latifur Khan. A machine learning approach to android malware detection. Intelligence and Security Informatics Conference (EISIC), 2012 European. IEEE, 2012.

[12] J. Kim, Y. Yoon, K. Yi, and J. Shin, SCANDAL: Static analyzer for detecting privacy leaks in Android applications. Security Informatics Conference, 2012 Available at http://mostconf.org/2012/papers/26.pdf

[13] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: scalable and accurate zero-day android

malware de-tection.In Proceedings of the 10th international conference on Mobile systems, applications, and services, MobiSys 12, 2012.

[14] Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In Proc. of IEEE Symposium on Security and Privacy, pages 95–109, 2012.

[15] L.-K. et al. Droidscope: Seamlessly reconstructing os and dalvik semantic views for dynamic Android malware analysis.In Proc of USENIX Security Symposium, 2012.

[16] Sahs J, Khan L. A machine learning approach to Android malware detection. In: European Intelligence and Security Informatics Conference.

[17] T. Isohara, K. Takemori, and A. Kubota,Kernel-based behavior analysis for android malware detection pp. 1011–1015, 2011.

[18] W. Enck, et al. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 393–407, 2010

[19] Burguera, U. Zurutuza, S. and Nadjm-Tehrani, Crowdroid: behavior-based malware detection system for Android Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile devices (New York, NY, USA, 2011), SPSM 11, ACM, pp. 15–26.

[20] Aafer, Yousra, Wenliang Du, and Heng Yin. DroidAPIMiner: Mining API-level features for robust malware detection in android." Security and Privacy in Communication Networks. Springer International Publishing, 2013. 86-103.

[21] Martinelli, Fabio, Andrea Saracino, and Daniele Sgandurra. Classifying Android Malware through Subgraph Mining. Revised Selected Papers of the 8th International Workshop on Data Privacy Management and Autonomous Spontaneous Security-Volume 8247. Springer-Verlag New York, Inc., 2013.

[22] Xie, Liang, et al. pBMDS: a behavior-based malware detection system for cellphone devices. Proceedings of the third ACM conference on Wireless network security. ACM, 2010.

[23] Wu, Wen-Chieh, and Shih-Hao Hung. DroidDolphin: a dynamic Android malware detection framework using big data and machine learning. Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems. ACM, 2014.

[24] Cheng, Jerry, et al. Smartsiren: virus detection and alert for smartphones. Proceedings of the 5th international conference on Mobile systems, applications and services. ACM, 2007.

[25] Zheng, M., Sun, M., & Lui, J. (2013, July). Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on (pp. 163-171). IEEE.

[26] Rastogi, Vaibhav, Yan Chen, and Xuxian Jiang. Droidchameleon: evaluating android anti-malware against transformation attacks. Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. ACM, 2013.

[27] Jayadeva. Learning a hyperplane classifier by minimizing an exact bound on the VC dimensioni. NEUROCOMPUTING 149 (2015): 683-689.

[28] Christodorescu, M., Jha, S., Seshia, S., Song, D., & Bryant, R. E. (2005, May). Semantics-aware malware detection. In Security and Privacy, 2005 IEEE Symposium on (pp. 32-46). IEEE.

[29] Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P. G., & Alvarez Maranon, G. (2013). MAMA: manifest analysis for malware detection in android Cybernetics and Systems, pg 469-488.

[30] Wagner, Cynthia, et al.Malware analysis with graph kernels and support vector machines. Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on. IEEE, 2009.

[31] Cristianini, Nello, and John Shawe-Taylor. An introduction to support vector machines and other kernel-based learning methods. Cambridge university press, 2000.

[32] Vapnik, Vladimir N. An overview of statistical learning theory. Neural Networks, IEEE Transactions on 10.5 (1999): 988-999.

[33] Joachims, Thorsten. Text categorization with support vector machines: Learning with many relevant features. Springer Berlin Heidelberg, 1998.

[34] Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., & Bringas, P. G. (2010). Idea: Opcode-sequence-based malware detection. In Engineering Secure Software and Systems (pp. 35-43). Springer Berlin Heidelberg.

[35] Lee, J., Jeong, K., & Lee, H. (2010, March). Detecting metamorphic malwares using code graphs. In Proceedings of the 2010 ACM symposium on applied computing (pp. 1970-1977). ACM.

[36] Hido, Shohei, and Hisashi Kashima.A linear-time graph kernel. Data Mining, 2009. ICDM09. Ninth IEEE International Conference on. IEEE, 2009.

[37] Aung, Zarni, and Win Zaw. Permission-based Android malware detection.International Journal of Scientific and Technol-ogy Research 2.3 (2013): 228-234.

[38] Shang, Zeng,Xu. Detecting Malware Variants thru Function Call Graph Similarity, 5th International Conference on Ma-licious and Unwanted Software, 2010

[39] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, July 1999. [40] RiskIQ, Feb 19 2014, Research Also Shows Steady and Significant Drop in Number of Malicious Apps Being Removed in Past Three Years. Available: http://www.riskiq.com/company/pressreleases/riskiqreports-malicious-mobile-apps-google-

[40] play-have-spiked-nearly-400

[41] Online Available at, http://virustotal.com

[42] Trendmicro mobile security. Available at http://www.trendmicro.com/us/enterprise/productsecurity/mobile-security/.

[43] M. Parkour. Contagio Mobile. Mobile Malware Mini Dump. http://contagiominidump.blogspot.com [44] Genome Project. Android malware samples. http://www.malgenomeproject.org.