INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 3, Issue 6)
Available online at www.ijariit.com

Android Malware Detection through Online Learning

Vinu Thadevus Williams
Computer Science and Engineering
College of Engineering Kidangoor, Kottayam, Kerala
vinuwilliams2016@gmail.com

Dr. K. S. Angel Viji
Computer Science and Engineering
College of Engineering Kidangoor, Kottayam, Kerala
angelhevin@yahoo.com

***Abstract**: Android malware constantly evolves so as to evade detection. The entire malware population to be non-stationary. Contrary to this fact, most of the prior works on machine learning based android malware detection have assumed that the distribution of the observed malware characteristics (i.e., features) does not change over time. The problem of malware population drift and propose a novel online learning based framework to detect malware, named CASANDRA (Context-aware, Adaptive and Scalable Android malware detector). In order to perform accurate detection, a novel graph kernel that facilitates capturing apps security-sensitive behaviours along with their context information from dependence graphs is proposed. Besides being accurate and scalable, CASANDRA has specific advantages: first, being adaptive to the evolution in malware features over time; second, explaining the significant features that led to an apps classification as being malicious or benign.*

***Keywords**: Online Learning, Malware Detection, Graph Kernels, Concept Drift.*

I. INTRODUCTION

Malware detection for mobile platforms such as Android has evolved as one of the challenging problems in the field of cyber-security. The number of new Android malware applications (apps) has grown tremendously in recent years. Machine Learning based malware detection: Machine Learning (ML) techniques have been predominantly used to perform malware detection on various platforms (such as Windows and Android). This is because, ML methods automatically learn the characteristics that distinguish malicious behaviour, when trained using a collection of malware and benign samples making them amenable to automated detection. ML based approaches extract semantic features from apps behaviours and apply standard classification algorithms (e.g., Support Vector Machine (SVM), Random Forests (RFs), etc.) to perform binary classification. These approaches typically use features such as system calls/Application Programming Interfaces (APIs) invoked, resources and privileges used, control- and data-flows inside apps execution to detect malicious behaviour patterns. These semantic features are extracted through static and dynamic program analysis.

Using Graph Representations Malware Detection

Malware Variants: A major reason for the tremendous growth rate in malware is the production of malware variants. The attackers produce a large number of variants of the same malware by resorting to techniques such as variable renaming and junk code insertion. These variants perform same malicious functionality, with apparently different syntax, thus evading syntax-based detectors. Higher level semantic representations such as call graphs, control, and data-flow graphs, and program-dependency graphs mostly stay similar even when the code is considerably altered. Program Representation Graph (PRG) to refer. ML classifiers are readily applicable on data represented as vectors and attempt to encode PRGs as feature vectors.

Kernels and Graph Mining: Shelf graph mining algorithms on PRGs for malware detection. Many classic graph mining based approaches are NP hard and have severe scalability issues, making them impractical for real-world malware detection. One of the increasingly popular approaches in ML for graph-structured data is the use of graph kernels. These graph kernels could be used together with a kernel classifier (e.g., SVM) to perform graph classification. These kernels have been known to operate in linear-time and have produced accurate results in real-world applications.

ML Based Malware Detection Challenges

All ML based Android malware detection techniques operate in a batch-learning setting and use off-the-shelf batch learners like SVM or RFs. The detection model is built using a batch of labelled benign and malware samples and is subsequently used to predict whether a given new sample is benign or malicious.ML based approaches are typically plagued by four challenges that make them unsuitable for large-scale real-world malware detection:

(C1) Population Drift: Though batch-learning based solutions are promising, their success is predicated on an important assumption that may not hold for the malware detection problem. Assume that the malware population (i.e., training data) used to build the detection model does not change over time. However, malware does not fit this profile.
(C2) Volume: In order to keep abreast with drifting population, batch learners have to be frequently retrained with huge volumes of data. Hence they pose severe scalability issues when used in the Android malware detection context where thousands of apps streaming in every day.
(C3) Explain Ability: ML based solutions just predict the labels of a given sample without offering insights or explanations into how those predictions arrive. In other words, they act as black-box solutions. However, for malware detection models, understanding the reasons behind their predictions is important in assessing their trustworthiness.
(C4) Expressiveness: PRGs are known to be complex and expressive data structures that characterize topological relationships among program entities. In many cases such representations fail to capture all the vital information from PRGs, thus losing their expressiveness. Unfortunately, the above-mentioned methods which capture structural information well, fail to capture the contextual information and this leads to raising false alarms even when sensitive operations are performed with users' consent. The general purpose graph kernels such as also suffer from the same drawback.

Proposed Approach

1. Accuracy: Accuracy of CASANDRA, which is a PRG based approach depends on how well it retains PRG's expressiveness. Contextual Weisfeiler-Lehman kernel (CWLK) that is specifically designed to capture both structural and contextual information from PRGs.
2. Efficiency: CASANDRA achieves its efficiency through the combined use of a scalable graph kernel (i.e., CWLK) and a state-of-the-art online classifier, namely, Confidence Weighted (CW) algorithm
3. Adaptiveness: CASANDRA automatically adapts to malware population drift through its use of online classifier.
4. Explain ability: Since CASANDRA uses a linear classifier along with CWLK which permits explicit feature vector representation of PRGs, we are able to categorically identify the PRG sub graph features which contribute to its predictions.
Developed CWLK, a graph kernel that is specifically designed to perform malware detection using PRGs. CWLK captures both contextual and structural information, enabling it to achieve high accuracy in a batch learning setting. DroidOL used a general purpose kernel which can only capture structural information from PRGs.

Three Following New Contributions in CASANDRA

I. Performing explainable malware detection is a unique feature of CASANDRA. As we demonstrate through our experiments, CASANDRA's explanations are more comprehensive and semantically closer to the malicious behaviours than those of state-of-the-art approaches.
II. Adaptiveness is another important trait of CASANDRA which is not prevalent in any existing approach. We have designed new experiments to thoroughly demonstrate how CASANDRA adapts to malware population drift (see Section V-D). Though DroidOL possessed this quality, it was neither experimentally verified nor demonstrated.
III. We also replaced the online learner used in DroidOL with a more recent state-of-the-art online learner. This resulted in significantly better accuracies.

II. LITERATURE SURVEY

1) Weisfeiler-Lehman Graph Kernels
Here propose a family of efficient kernels for large graphs with discrete node labels. Key to our method is a rapid feature extraction scheme based on the Weisfeiler-Lehman test of isomorphism on graphs. It maps the original graph to a sequence of graphs, whose node attributes capture topological and label information. A family of kernels can be defined based on this Weisfeiler-Lehman sequence of graphs, including a highly efficient kernel comparing subtree-like patterns. Its runtime scales only linearly in the number of edges of the graphs and the length of the Weisfeiler-Lehman graph sequence. In experimental evaluation, our kernels outperform state-of-the-art graph kernels on several graph classification benchmark data sets in terms of accuracy and runtime. Kernels open the door to large-scale applications of graph kernels in various disciplines such as computational biology and social network analysis.
Keywords: graph kernels, graph classification, similarity measures for graphs, Weisfeiler-Lehman algorithm

2) Confidence-Weighted Linear Classification
Confidence-weighted linear classifiers, which add parameter confidence information to linear classifiers. Online learners in this setting update both classifier parameters and the estimate of their confidence. The particular online algorithms we study here maintain a Gaussian distribution over parameter vectors and update the mean and covariance of the distribution with each instance. Empirical evaluation on a range of NLP tasks shows that algorithm improves over another state of the art online and batch methods, learns faster in the online setting and lends itself to better classifier combination after parallel training.

3) Empirical Assessment of Machine learning-based Malware Detectors for Android
To address the issue of malware detection through large sets of applications, researchers have recently started to investigate the capabilities of machine-learning techniques for proposing effective approaches. So far, several promising results were recorded in the literature, many approaches being assessed with what we call in the lab validation scenarios. The purpose of malware detection to discuss whether such in the lab validation scenarios provide reliable indications on the performance of malware detectors in real-world settings, aka in the wild. To this end, Several Machine Learning classifiers that rely on a set of features built from applications' CFGs. use a sizeable dataset of over 50 000 Android applications collected from sources where state-of-the art approaches have selected their data. In the lab, approach outperforms existing machine learning-based approaches.

4) Adaptive and Scalable Android Malware Detection through Online Learning
It is well-known that malware constantly evolves so as to evade detection and this causes the entire malware population to be non-stationary. Contrary to this fact, prior works on machine learning based Android malware detection have assumed that the distribution of the observed malware characteristics (i.e., features) do not change over time. Address the problem of rnalware population drift and propose a novel online machine learning based framework, named DroidOL to handle it and effectively detect malware. In order to perform accurate detection, the security-sensitive behaviour is captured from apps in form of inter procedural control flow sub-graph features using a state-of-the-art graph kernel. In order to perform scalable detection and to adapt to the drift and evolution in mal ware population, an online passive-aggressive classifier is used.

5) Contextual-Weisfeiler-Lehman Graph Kernel for Malware Detection
A novel graph kernel specifically to address a challenging problem in the field of cyber-security, namely, malware detection. Previous research has revealed the following: (1) Graph representations of programs are ideally suited for malware detection as they are robust against several attacks, (2) Besides capturing topological neighbourhoods (i.e., structural information) from these graphs it is important to capture the context under which the neighbourhoods are reachable to accurately detect malicious neighbourhoods. Observe that state-of-the-art graph kernels, such as Weisfeiler-Lehman kernel (WLK) capture the structural information well but fail to capture contextual information. Develop the Contextual Weisfeiler-Lehman kernel (CWLK) which is capable of capturing both these types of information. The malware detection problem, CWLK is more expressive and hence more accurate than WLK while maintaining comparable efficiency.

III. PROPOSED METHOD

BACKGROUND AND MOTIVATION
Why considering structural information alone from PRGs is insufficient to determine the maliciousness of a sample and how supplementing it with contextual information helps to increase the detection accuracy.

Why using batch learning is impractical for building a real-world malware detector and how the use of online learning alleviates such practicality issues.

Motivations for CWLK

To demonstrate the necessity of CWLK, we use a real-world Android malware from the Geinimi family which steals user's private information and contrast its behaviour with that of a well-known benign app, Yahoo Weather. Geinimi's execution: The app is launched a background event such as receiving an SMS or call. Once launched, it reads the user's personal information such as geographic location and contacts and leaks the same to a remote server. The method leak location reads the geographic location through getting Latitude and gets Longitude APIs. Subsequently, it calls a leak_info_to_url method to leak the location details (through DataOutputStream.writeBytes) to a specific server. The nodes in ADG are labelled with the sensitive APIs that they invoke and the edges denote the control-flow dependencies. Yahoo Weathers execution: On the other hand, Yahoo Weather could be launched only by user's interaction with the device (e.g., by clicking the app's icon on the dash board). The app then reads the users location and sends the same to its weather server to retrieve location-specific weather predictions. Hence, ADG portions of Yahoo Weather is same as that of Geinimi. Geinimi malicious is the fact that its leak happens without the user's consent. In other words, unlike Yahoo Weather, Geinimi leaks private information through an event which is not triggered by user's interaction. We refer to this as a leak happening in user-unaware context. On the same lines, we refer to Yahoo Weathers leak as happening in user-aware context.in the case of Android apps, one could determine whether a PRG node is reachable under user-aware or user-unaware context by examining its entry point nodes. Following this procedure, we add the context as an attribute to every ADG node.

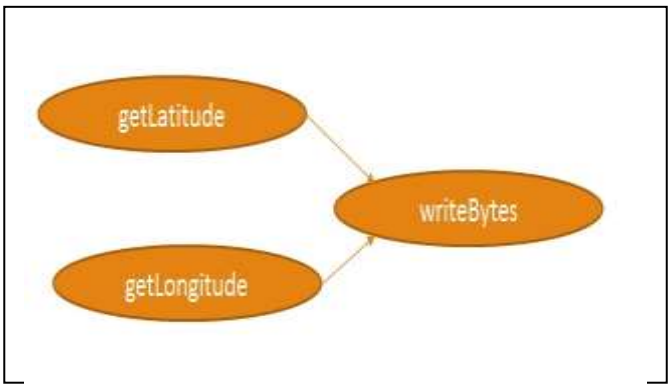


Figure 1: API Dependency Graph

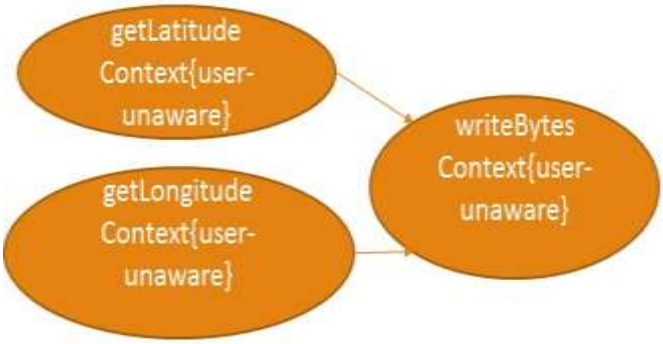


Figure 2: Example of Malware Application Gemini Application

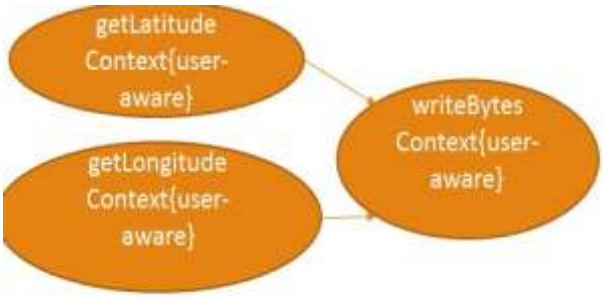


Figure 3: Example of Benign Application Yahoo Whether Application

(R1) Capturing structural information: Since malicious behaviours often span across multiple nodes in PRGs, just considering individual nodes (and their attributes) in isolation is not enough. For instance, in the case of Geinimi, the privacy leak attack spans across three ADG nodes. Therefore, capturing the structural (i.e., neighbourhood) information from PRGs is of paramount importance.

(R2) Capturing contextual information: Considering just the structural information without the context is not enough to determine whether a sensitive behaviour is triggered with or without users' knowledge. For instance, if structural information alone is considered, the features of both Geinimi and Yahoo Weather apps become identical, thus making the latter a false positive. Hence, it is important for the detection process to capture the contextual information as well to make it more accurate.

Many existing graph kernels could address the first requirement well. However, the second requirement which is more domain-specific makes the problem particularly challenging. To the best of our knowledge, none of the existing graph kernels support capturing this reachability context information. In summary, this gives us a clear motivation to develop a new kernel that specifically addresses our two-fold requirement.

Motivations for Using Online Learning

(1) Handling population drift.

(2) Handling large volumes of high dimensional data.

(3) Performing detection on data that streams-in at real-time.

Malware detection-specific justifications on how online learning helps to address these challenges are presented below.

Handling Population Drift: attacks that were popular at some point in time could not sustain forever. For instance, the Base Bridge family of malware leveraged on two root exploits, namely, RATC and Zimperlich to escalate its privileges Once the corresponding vulnerabilities were patched and the AV vendors became capable enough to detect such attacks, Base Bridge's propagation and sustenance were affected, ultimately resulting in its extinction. This leads to the following observation: several malware families emerge, flourish and fade-away over time due to various domain-specific reasons. From an ML based malware detection viewpoint, this leads to emergence, dominance, and disappearance of semantic features that characterize these attacks and evasion strategies. This results in the three following phenomena that happen overtime:

(1) New features emerge

(2) The significance of features Vary

(3) The cumulative number of features keeps monotonically increasing.

This causes the underlying distribution of malware samples to change over time leading to what is known as a concept drift. As a result of this drift, the predictions of a static model might become less accurate in due course of time. Therefore, detectors need to adapt to such changes quickly and accurately. A straightforward solution to this problem is to detect the magnitude and direction of drift in the concept over time and retrain the batch models when the drift is significant. However, this approach would be hampered if: (1) the data arrives in large volumes at a rate too fast to detect (2) retraining the models at regular intervals is too expensive. Unfortunately, both these conditions are true in the real-world malware detection setting, as we review in detail below. On the other hand, online learners which are trained on a stream of samples as they arrive are naturally adaptive to evolving distributions mainly due to their learning mechanisms that continuously and efficiently update the model with the most recent examples. Handling large high-dimensional data: Particularly in the malware detection problem, data is large not only in sample size but also in feature size. For instance, on the large-scale dataset used in our evaluation, even a light-weight method such as DREBIN extracts and uses more than 1 million semantic features. Many batch learners (e.g., SVMs, RFs) typically take multiple passes over the training samples to optimize their parameters and yield the most accurate models within their hypothesis space. Handling real-time streaming: Android has a vibrant and growing app ecosystem with a considerable number of third-party markets. Google Play, the official market, host more than 2.4 million apps as of this writing.

IV. METHODOLOGY

The methodology of CASANDRA framework designed to perform context-aware, scalable, adaptive and explainable malware detection is presented in this section

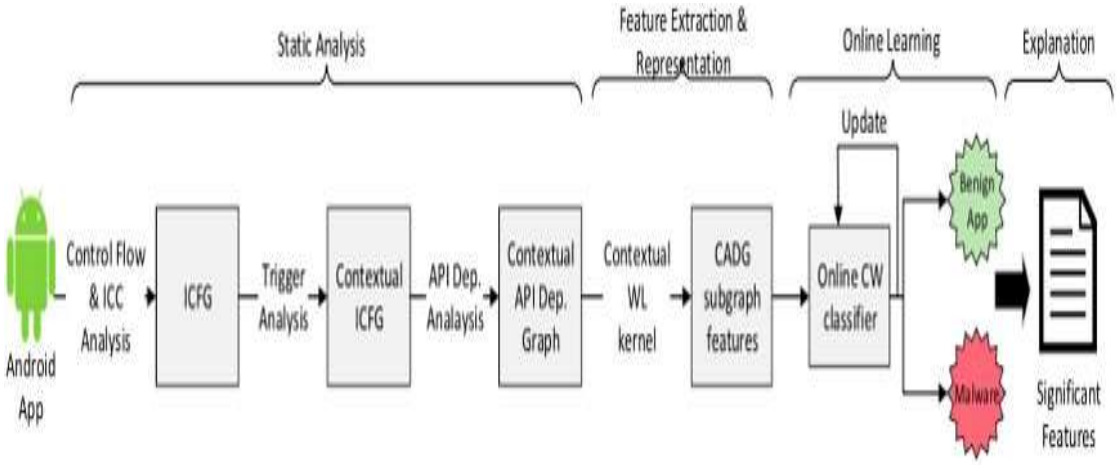


Figure 4: Casandra Framework Overview

Casandra Framework Overview

Static Analysis: Our framework considers sub graphs from Contextual API Dependency Graphs (CADGs) as semantic features to perform malware detection. To this end, we first perform static program analysis to transform the given set of apps into their corresponding CADG representations.

Figure 5: Static Analysis

Feature Extraction & Representation: Once the CAGD of the apps in the dataset is constructed, those sub graphs which represent the security-sensitive events that happen in every app along with their context are extracted as using CWLK. We follow a Bag-of-Features (BOF) model to construct the feature vector of individual apps.

Online Learning: Once the feature vectors of all the apps in the training set are built, we train a CW classifier with them to detect malware. CW classifiers training and update procedures are explained based malware detection solutions are usually black-box methods as they do not explain why a particular sample is detected as malicious or benign. CASANDRA reports significant CADG sub graphs of an app that contribute to a detection.

Static Analysis

CASANDRA considers CADG sub graphs as features which encompass both structural and contextual information characterizing security sensitive operations from apps. We perform a comprehensive static analysis on a given app to construct its CADG. Our static analysis and CADG construction process are similar to that of Droid SIFT and App Context.

Step 1: Inter-procedural Control Flow Graph (ICFG) Construction: the nodes of ICFG are the instructions in every method. The ICFG edges are the intra- and inter-procedural control flows among these instructions. There two types of interprocedural invocations in Android apps:

(1) Method invocation
(2) Inter Component Communications (ICCs).

Method invocations could be directly inferred through control flow analysis. However, ICCs are not straightforward as these invocations are facilitated by the underlying Android OS framework. In order to model such invocations, we leverage on the IC3 tool [41] and perform a precise ICC analysis.

Step 2: Contextual ICFG (CICFG) Construction: During the course of execution of the program, the ICFG nodes could be reached by actions triggered by entities such as user or system. Trigger events are the external events such as users' interaction with app's user-interface (UI) and system changes such as receipt of SMS that trigger invocation of security-sensitive APIs. This procedure and categorize trigger events of ICFG nodes into the three following types:

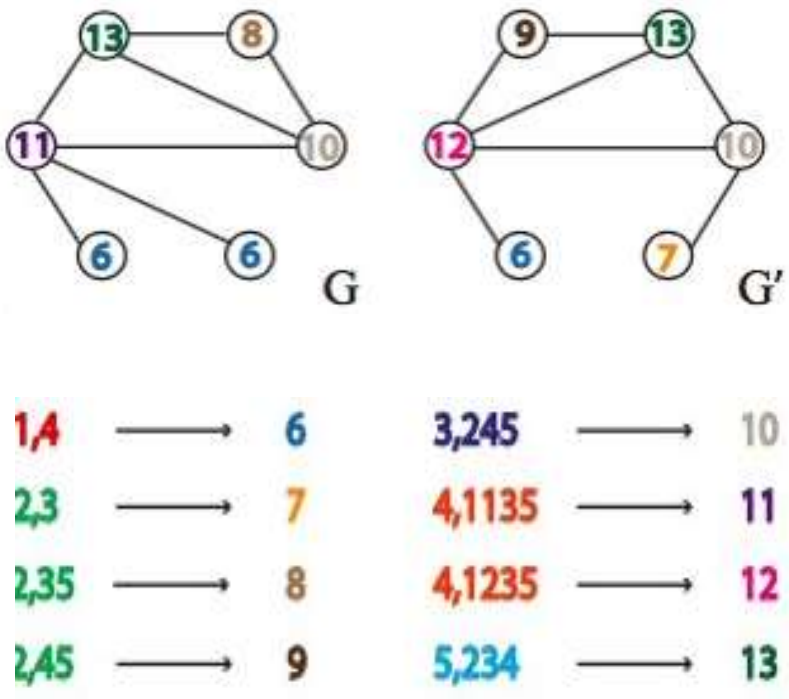
1) UI triggers: events triggered by interactions on app's UI (e.g., Clicking UI buttons to make calls, etc.).

2) NON-UI triggers: events initiated by the system state changes (e.g., receipt of SMS, change of signal strength), and events initiated by hardware related actions (e.g., pressing the HOME or BACK button).

3) UNRESOLVED triggers: Entry points of certain sensitive methods are not traceable by our static analysis. For instance, DroidKungFu, a popular malware family has its malicious payload triggered by dynamically loaded code.

Definition 1 (CICFG): CICFG = (Ni,Ei, ξ) of an app a is a directed graph in which each node n ∈ Ni denotes an instruction in every method of a, and each edge e(n1, n2) ∈ Ei denotes either an intra-procedural control-flow dependence from n1 to n2 or a calling relationship from n1 to n2 . ξ is a set of contexts though which every node n ∈ N could be reached.

Step3: CADG construction: The CICFG captures the complete control-flow across every instruction in an app, along with context information. CADG of an app is obtained from its CICFG by considering only the nodes that access security-sensitive APIs4 along with their context information.



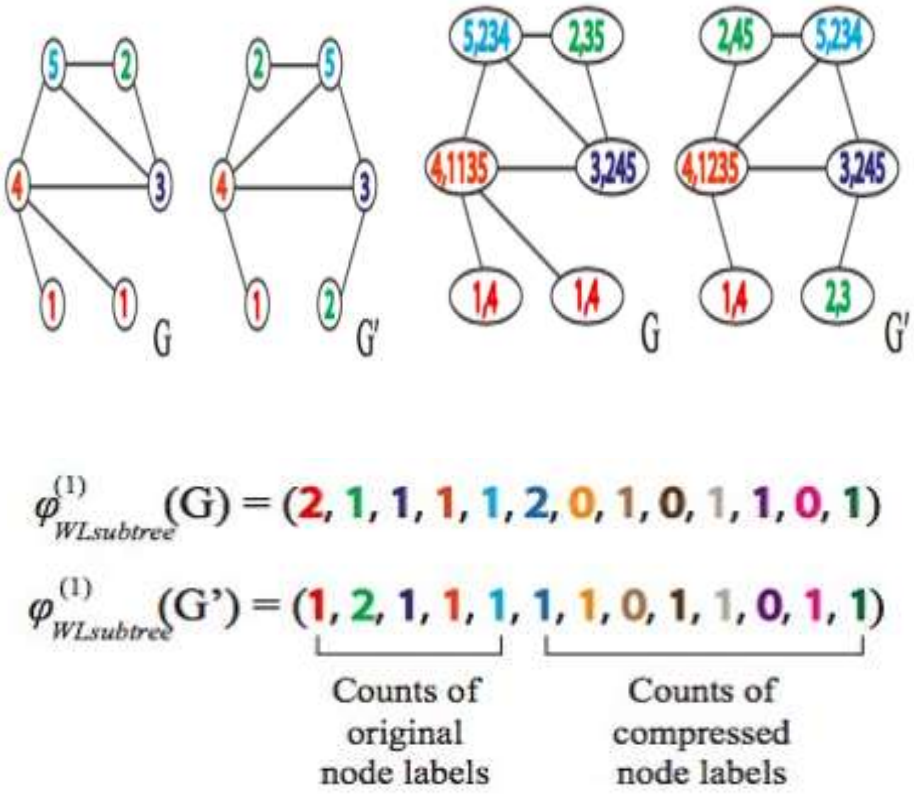


Figure 6: WLK Algorithm

Definition 2 (CADG): A CADG can be represented as a 4- tuple CADG = (N, E, λ, ξ), where N is a finite set of nodes and n ∈ N is an instruction of a method that corresponds to invoking a security-sensitive API and therefore N ⊆ Ni. E ⊆ N × N is a set of directed edges where an edge from e(n1, n2) ∈ E exists iff there exists a path p(n1, n2) between these two nodes in the CICFG and method(n1) = method(n2), where method(n) denotes the method that encompasses instructionn.5 λ is the set of labels representing the security-sensitive APIs and N → λ is a labeling function which assigns a label to each node. ξ is a set of triggers though which every node in the CADG could be reached and C → ξ is a function which assigns the context to each node. In both these apps, all three CICFG nodes invoke security-sensitive APIs and they are retained in CADGs. All of these nodes belong to the same method (i.e., leak_info_to_url) and hence the path that existed among them in CICFG translate to edges in CADG.

Feature Extraction & Representation Using CWLK

Once the CADGs are constructed, our next task is to extract semantic features that characterize sensitive behaviors of apps from them. To this end, we use CWLK, a graph kernel we developed in our previous work which is specifically designed to capture both structural and contextual information from PRGs and perform accurate malware detection. This directly addresses the requirements R1 and R2.

1: Multiset-label determination
- For $i = 0$, set $M_i(v) := l_0(v) = \ell(v)$. [2]
- For $i > 0$, assign a multiset-label $M_i(v)$ to each node v in G and G' which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.

3: Label compression
- Sort all of the strings $s_i(v)$ for all v from G and G' in ascending order.
- Map each string $s_i(v)$ to a new compressed label, using a function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G and G'.

We furnish the details on how CWLK supports CASANDRA in extracting CADG sub graph features to perform malware detection, in this subsection. For detailed discussions on CWLK use cases with other PRGs (e.g., ICFG), comparison with WLK and other graph kernels. We begin by explaining the regular WLK, then introduce CWLK and finally discuss how WLK falls short when performing malware detection using CADGs and how CWLK rectifies the same. Weisfeiler-Lehman Kernel: WLK, works by decomposing graphs into sub graphs in such a way that a kernel function for a pair of graphs can be defined as a convolution of kernel functions defined over their sub graphs.

Online Learning: Once the feature vectors of all the apps in the training-set are built using CWLK, we train an online CW classifier with them to detect malware. CW classifier's training and update procedures are as explained below with relevant notations. Denote the features of an app (both benign and malware) as a vector x = [x (1), x (2). . . x(d)] T, and its label as y ∈ {−1, +1}, where −1 indicates benign and +1 indicates malicious apps. The CW classifier receives a number of samples, xi, and their labels, yi, and trains using this labeled data. Given a new unseen sample, x, the goal of CW classifier is to predict the label, y, of this new sample based on its trained model. CW classifier fits a linear decision boundary (i.e., hyper plane) between the positive and negative class samples. That is, the model is a weight vector, w = [w (1), w (2). . . w(d)]T which indicates the weight (i.e., relative importance) of each of the features used to predict the output label y. The predicted label, ˆy, is the sign of the inner product between x and w:

$\hat{y} = \text{sign}(x, w)$

V. CONCLUSION

CASANDRA, an online learning based Android malware detection framework. CWLK, a novel graph kernel that facilitates capturing apps' security-sensitive behaviors along with their context information from dependency graphs is also proposed. CWLK supports explicit feature vector representation of apps' dependency graphs using which an online classifier is trained to detect malware. Our large-scale evaluations on both recent real-world and benchmark datasets demonstrate that CASANDRA outperforms two state-of-the-art techniques. This superior performance and scalability make CASANDRA, in particular, and online learning based solutions, in general, better candidates for the malware detection task.

Future Work: We plan to investigate three specific directions in our future work:

1) Taking into account the recent developments in the area of Deep Graph Kernels which show potentials to learn latent sub-structures from graphs to achieve better accuracy, we intend to explore on the deep learning variant of CWLK in our future work.

2) We plan to incorporate contextual information in other sub-structure based graph kernels such as NHGK and NSPDK and subsequently, investigate their suitability for malware detection.

3) API dependencies used in CASANDRA represent only one perspective (i.e., view) of the apps. However, as revealed by recent works such as MADAM and Reveal Droid capturing other views such as information flows, and permission dependencies lead to more comprehensive detection. Inspired by this inference, we intend to extract multiple features sets from PRGs and systematically integrate those using Online Multiple Kernel Learning methods, for performing a more comprehensive malware detection.

Dataset Release: To allow easy reproduction and verification of our work, we provide all the datasets used within this work for download at https://sites.google.com/view/casandrantu. Here we analyse the system obtain the efficiency up to 60 %. In this features analyses a system to develop a virus free android phone. Applying the ranking system based on context and batch form. SVM & Robert forest learning algorithm combined to execute increase the accuracy of the learning process.

REFERENCE

1. A. Narayanan et al., "Contextual Weisfeiler-Lehman graph kernel for malware detection," in Proc. Int. Joint Conf. Neural Netw., 2016, pp. 4701–4708.
2.Symantec 2016 Threat Report. [Online].Available:https://www.symantec.com/content/dam/symantec/docs/reports/istr-21–2016-en.pdf.
3. M. Zhang et al., "Semantics-aware android Malware classification using weighted contextual API dependency graphs," in Proc. ACM SIGSAC Conf. Comput. Commun. Secure, 2014, pp. 1105–1116.
4. M. Dredze et al., "Confidence-weighted linear classification," in Proc. 25th Int. Conf. Mach. Learn., 2008, pp. 264–271.
5. N. Shervashidze et al., "Weisfeiler-Lehman graph kernels," J. Mach. Learn. Res., vol. 12, pp. 2539–2561, 2011.

