2014 IEEE 28-th Convention of Electrical and Electronics Engineers in Israel

Adware Detection and Privacy Control in Mobile Devices

Ianir Ideses
Shine Security
ianir.ideses@getshine.com

Assaf Neuberger
Shine Security
assaf_ne@getshine.com

***Abstract*—In this paper we propose a system and algorithms for detection of Adware in mobile devices that are based on machine learning algorithms and are capable of adapting to the ongoing transformation of Adware and Malware. The system is based on static and dynamic analysis of mobile applications, extraction of useful features and real-time classification. This classification is based on supervised machine learning algorithms with emphasis on fast, linear operations and efficient implementation on mobile devices.**

The system presented in this paper enables identification of relevant features that are salient in Adware and Malware, useful for further analysis by security researchers. The proposed system exhibits a detection rate of 97%. The system has been tested and verified on known, industry standard, datasets and is superior to state of the art solutions available in the market. These result have been verified by 3rd party evaluators.

***Index Terms* - Android, SVM, Adware, Malware, AV, Static and Dynamic Analysis**

I. Introduction

The Android operating system (OS) is quickly becoming the dominant OS for mobile devices. One of the major driving forces of this OS is the ability to develop and publish 3rd party applications with a very low entry barrier. Applications built for the Android OS can be monetized in several ways, among them are paid applications, freemium applications and applications that rely on ads for revenue. Among these 3 models, ad based applications form a significant part of Google's application market (Play Store). With the growing market for ad based applications, another type of market emerged, that of advertisement SDKs (Software Development Kits). These SDKs enable application developers to incorporate ad revenue streams into their applications with relative ease. It is this ease that sparked a new form of application, named Adware. Adware are essentially applications whose main purpose is getting maximal revenue to the developer while giving the user the minimal amount of value. Sometimes, giving the user no value at all.

These Adware may cause the user harm by stealing his data and sending it to a remote server (for later use in ad targeting), by displaying ads aggressively via screen hijacking, by displaying ads in the notification area which is usually reserved for important system events and, in extreme cases, an Adware may hijack the device speaker. Once an Adware has been installed, it is not always easy to trace back the malicious behaviour on the phone to the corresponding application.

In order to address these issue of malicious software, several companies have begun developing anti-Malware software. In principle, all major vendors of desktop anti-Malware software have begun offering mobile versions of their software. In academia, research of anti-Malware algorithms has also become widespread. Algorithms for detection of Malware can be found in the works of [5], [7], [6] to name a few. These algorithms and systems usually rely on heuristics and signature files and are ill equipped to deal with new specimens of Malware and Adware. Other research is focused at using machine learning algorithms for automatic classification of Malware, example of such work can be found in [1], [2] and [4].

However, to date, no algorithm or system has been tailored to perform detection and classification of Adware by taking advantage of their core behaviour - that of delivering ads to the device. This behaviour is very specific and can not be easily obfuscated. Moreover, by utilizing standard SDKs, the authors of Adware leave a trace of their intentions. In this paper we describe a systematic algorithm that is capable of detecting Adware by automatically identifying the relevant features and is not limited to a set of signatures. This algorithm can also be trained to detect Malware as well as Adware and has shown a significant advantage over other commercial products.

Our sampling method for data specimens enables us to reveal *clusters*, namely group representatives of applications which are similar but not identical. Consequently, we can test the true generalization capabilities of our solution.

II. Adware detection

Detection of malicious software for mobile devices using machine learning techniques, be it Malware or Adware, requires several stages. These stages are described in the subsections below:

A. Definition of training and testing sets

Defining what constitutes an Adware, a Malware or a Goodware (benign software) is a critical stage in building a classifier. This is not a simple question, as there are no agreed upon definitions for Malware/Adware. Even software that is

unanimously agreed upon as Goodware may exhibit Malware-like behaviour. In order to tackle this problem, several methodologies and heuristics were used.

1) Data repositories: The first stage in defining a set of Malware, Adware and Goodware is acquisition. There are several known standard repositories that have already been labeled, among them are [2], [8]. In addition, our security experts gathered specimens from various security mailing lists and user groups.

2) Data labeling: It is possible to query VirusTotal [9], a database maintained by Google, that contains sample labels which were assigned by 51 different anti-virus (AV) solutions. Let us denote an AV software av declaration of specimen s as $R_{av}(s) \in \{0, 1\}$, where 0 and 1 indicate benign and malicious labels respectively. The *malrank* is simply defined as

$$R_{mal}(s) = \sum_{av} R_{av}(s) \tag{1}$$

Moreover, and AV declaration $R_{av}(s)$ is accompanied with a signature name which the AV software has provided for each sample. We denote the signature name as $S_{av}(s)$. The aforementioned names yield extremely valuable information, because various AV software tend to choose them similarly. For example, a specimen which is attributed as "Trojan" by many AV software, is probably a trojan-horse type of Malware. In the Adware realm, the name can expose the ad-network SDK which was utilized by the specimen, e.g., "Airpush". We gathered common Adware-related keywords into a set which will be denoted as K. The *ad-rank* $R_{ad}(s)$ of specimen s is the amount of names $S_{av}(s)$ which are included in K:

$$R_{ad}(s) = \sum_{av} |\{S_{av}(s)\} \cap K| \tag{2}$$

It should be noted that by definition,

$$R_{ad}(s) \leq R_{mal}(s) \tag{3}$$

The following rules describe our labeling policy:

- *Malware:* $R_{mal}(s) \geq N_1 \wedge R_{ad}(s) = 0$
- *Adware:* $R_{mal}(s) \geq N_2 \wedge R_{ad}(s) \geq N_3$
- *Goodware:* $R_{mal}(s) = 0$

for the labeling policy .

The thresholds N_1, N_2 and N_3 were set to 15, 5 and 5 respectively, chosen by consulting with our security experts. Our raw data set includes about 660,000 Malware, 394,000 Adware and 251,000 Goodware. The next subsection will describe how to prune almost identical specimens.

B. Removing pseudo-duplicates

Identifying almost duplicate samples is an important task for preventing an unnoticed *overfit* in our experiment. Namely, testing the algorithm on samples which are *almost* identical to samples in the learning set. The following methods for sample removal were employed:

1) Package name duplicates: A Google Play application keeps its package name throughout newer versions. We chose a single specimen representative for each package name in our dataset.

2) Code duplicates: An Android application package (APK) has various components. By comparing a hash calculation on the classes.dex file, we ignore the APK resources and other side-components, thus easily targeting samples with identical code implementation.

3) VirusTotal duplicates: It is common for two specimens s_1, s_2 to have an identical signature name S_{av}. Such a statement by an AV is a result of long security research efforts which we have harnessed for detecting almost duplicate specimens. Lets us define an undirected graph $G = (V, E)$, where $v_i \in V$ is a specimen i, and $w_{ij} \in E$ is an edge between specimens i and j. The value of w_{ij} is defined as the number of AV software which entitle specimens i and j with the same name. By thresholding each weight $w_{ij} \geq th_w$, we can prune or enrich E for reflecting weak or strong resemblance between specimens. In order to identify truly different specimens, performing a connected component analysis on G reveals Malware and Adware *clusters*. In this fashion, we can control the inner-resemblance of specimens within our database, which on one hand reflects the real-world scenario of code reuse among developers, but on the other hand constrains possible overfit which can yield deceiving results. Fig. 1 depicts the graph representation of our Malware set, which reduces $\|V\| = N$ specimens into C connected components. Each connected component C_i contains N_i specimens which are considered as pseudo-duplicates, hence $\sum_i N_i = N$.

Our goal is to generate the most representative Malware data set given a target size $M \leq N$. We employed the following sampling heuristic: Subsequent sampling without replacement of each connected component C_i, until reaching the target data set sample size M. As opposed to random sampling without replacement, our scheme represents a wide variety of connected components even when the data set is small.

C. Feature extraction

Once datasets of Malware, Adware and Goodware have been selected, the next stage of feature extraction can commence. The Android operating system enables extraction of various features of applications relatively easily, for example, information regarding application permissions, broadcast receivers, activities and can be obtained by issuing simple operating system calls. Additional information such as bytecode fragments, system-calls call graphs and dynamic behaviour can also be extracted by techniques such as decompilation and dynamic analysis of the application at runtime.

D. Building classifiers

Once a set of features has been selected, the next stage may commence, that of building classifiers. We used SVM (Support Vector Machines [3]) in order to train our classifiers. We employed supervised training by partitioning the training

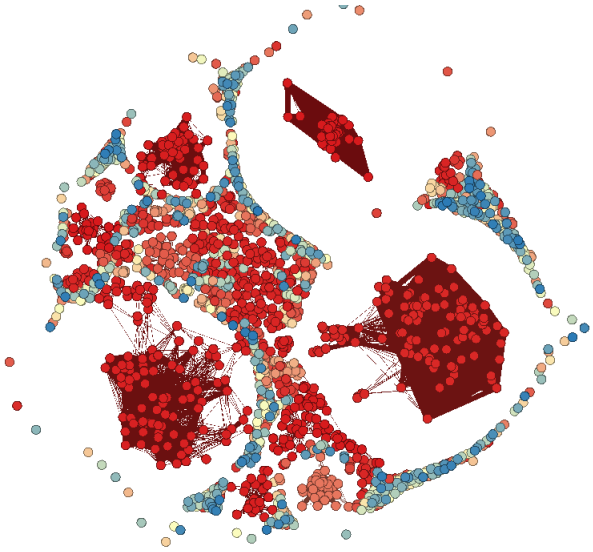

Fig. 1. Graph representation of malicious samples data set. Each node corresponds to a specimen, and each edge to the number of AV software which declare the specimen pair with the same signature name.

and test sets, and running training sessions using labels and features. Once training has completed, ROC (Receiver Operating Characteristics) graphs were obtained and a working point was selected. We chose to use linear SVM kernels due to three main reasons: The first is computational complexity. The classifier is designed to run on a mobile device which may be limited in terms of CPU, as well as memory resources. Choosing a linear kernel requires far less computations (a single dot product operation) and is simpler to implement. The second reason is the insights which the primal weight vector w contributes, as it can be seen as a weighting scheme for each feature which was included in the learning phase. Security experts can easily interpret and validate the resulting meaningful features.

For example, the following features obtained high weight in our learning scheme, while reflecting obvious ad-network affiliation:

- *com.mobclick.android.umengfeedback*
- *com.leadbolt.adnotification*
- *com.revmob.ads.fullscreen.fullscreenactivity*
- *com.airpush.android.pushservice*

The third reason for choosing a linear SVM classifier is its robustness to high-dimensional data. In our case, the dimensionality was of 12 million features. During training, irrelevant features receive relatively low weights in the primal weight vector, and will not impact the utilization of relevant features due to the globally optimal solution of the SVM. Moreover, a linear kernel copes with many dimensions without overfitting, unlike nonlinear kernels which increase data dimensionality for better separation in the learning set, but risk the classifier correctness in the test set.

E. Feature selection

Implementing the classifier on a mobile device introduces real world limitations. These include memory usage, CPU utilization, impact on battery consumption and storage usage. In order to reduce the amount of resources required for detection on the device, a feature selection process has been applied. This process consists of pruning of the features in a manner that takes into account the amount of resources and the discrimination abilities of each feature and reduces the amount of features so that the accuracy of the classifier is maintained as much as possible. This dimensionality reduction is performed while gaining significant reduction in the amount of resources. The result is a classifier that is suitable for deployment in a mobile device that has a limited amount of memory and CPU power.

Feature selection can be performed by solving the optimization problem as follows: We denote the resource requirements of a feature as:

$$Res(f) = \alpha * CPU + \beta * MEM \tag{4}$$

Where CPU is an estimation of the CPU usage required for extracting this feature and MEM is a measure of the memory usage required for the extraction. α and β are normalization constants. We define the cost of a feature by subtracting the resource value from the absolute value of the SVM coefficient:

$$Cost(f) = |C(f)| - \gamma * Res(f) \tag{5}$$

where $C(f)$ is the value of feature f in the resulting primal weight vector w, γ is a normalization constant and $Res(f)$ is the value as described in the previous equation.

The optimization problem is solved by greedily searching for the minimum cost for a given dimension size. Selection of the dimension is acquired by analysing the resulting ROC curves per dimensionality.

F. Reduction of false positives

In order to reduce the amount of false positives (FAR), ie: applications that are marked as Malware or Adware, but are benign, we coupled the score from the SVM classifier together with the following a-priori information: In Google Play, each application web page includes the approximated amount of user installations so far. Our specimens from Google Play represent approximately **30 billion** installations, whereas 9% of them were considered as Adware and only 0.05% as Malware. Evidently, Google Play is a very safe source for downloading an application as long as Malware are regarded, but where it comes to Adware - an Android user has high probability of encountering such applications. Under the assumption that Google play is the main source for Android applications, this paper reveals that Malware are not the main threat for Android users. Using this prior information, combined with an aggressive working point that was tuned to very low FAR, enabled us to reduce the amount of FAR to virtually 0%.

G. Implementation

Implementation of the classifier on the mobile device was carried out using the standard Android SDK. The majority of the code has been implemented in plain Java. In order to optimize the application, several techniques were used. The first is reduction of storage required for the classifiers. Storage required for the classifiers is dependant on the dimensionality of the feature vector and on the representation of the feature vector. Since most of the features are textual (e.g. permission names), we opted to hash them. After the strings have undergone a hashing function, the resulting string was cropped. The cropping factor was determined empirically so that no hash collisions were detected. In our case a length of 8 bytes was sufficient. This procedure enabled us to reduce the storage requirements of the application by a 90%. In addition to storage saving, features that required polling of data, for example listening on the system binder, were optimized to look in a time window after application installation and application launch instead of running constantly in the background. This enabled a massive reduction of resource utilization. Moreover, the classifiers were implemented as a *cascade*, where features whose resource toll is lower are computed first and used for classification. Subsequent stages of the cascade are triggered according to the classifier scores at each stage.

III. Testing methodology

Testing of our implementation was carried out both internally and by a 3rd party examiner who compared the detection performance of the proposed system to other commercial products.

A. Internal testing

The first experiment reviewed the detection of the Drebin malicious data set of 5500 specimens with a similar sized data set of Goodware specimens from our repository. The learn and test sets were chosen to be of equal size.

The second experiment explored the detection rate of Adware versus a similar sized set of Goodware. By using the pseudo-duplicates removal techniques described in section II-F, we have sampled a representative set of $16K$ Adware specimens. As before, learn and test sets were equally sized.

B. External testing

In order to test the detection performance of the proposed anti-Malware system, the binary file of the commercially available version of our system has been given to Capitol College [10], a leading security tester for benchmarking. Capitol College's uses its own proprietary databases which were not supplied to us. These included a set of more than 7000 samples.

IV. Results

A. Internal testing

Applying our most accurate classifier on the Drebin set, mixed with Goodware specimen yielded the following ROC curve, as seen in Fig. 2. After tuning the classifier cascade for an admissible working point, 99.3% of the malicious specimen were detected, with a 0.14% false-alarm rate.

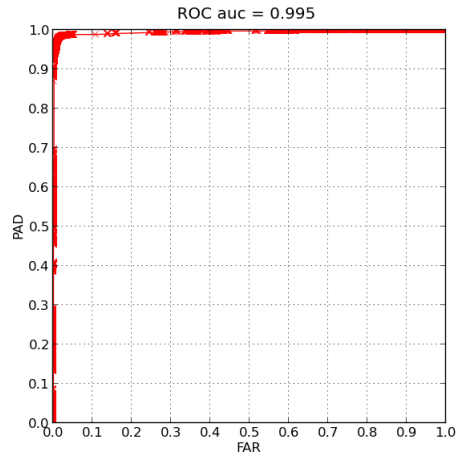

Fig. 2. ROC curve, Drebin vs. Goodware

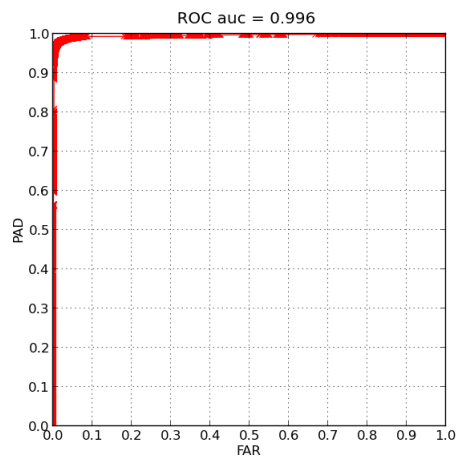

Fig. 3. ROC curve, Adware vs. Goodware

The Adware vs. Goodware ROC of our most accurate classifier is seen in Fig. 3. The cascade produced 89.1% detection rate, with a 0.1% false alarm rate.

B. External testing

Results can be seen in Fig. 4. The results indicate the superiority of our system to offerings by major anti-Malware vendors. As seen in the graph, our system outperforms all other major solutions by a wide margin. It is important to note that our system, unlike other commercial systems, is not dependent on signature files and is expected to continue presenting this high detection ratio for new, unknown samples, whereas competing systems are expected to require frequent updates in order to detect new samples.

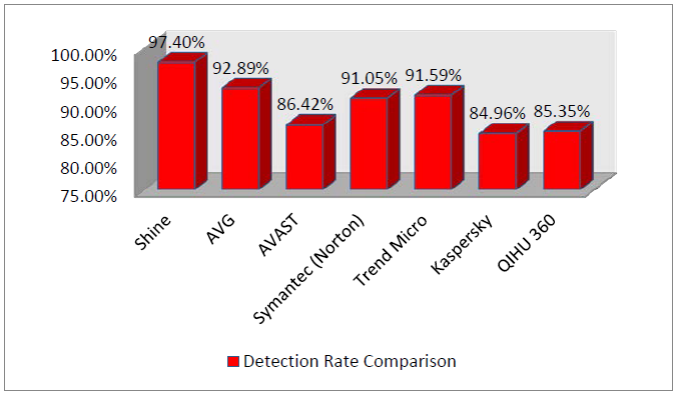

Fig. 4. Shine evaluation as received from Capitol College

V. Conclusion

In this paper we presented a novel system for detection of malicious software for the Android OS. The primary goal of the system was to target Adware, as they present similar and often predictable features which Adware authors use for monetization. In order to avoid the common pitfalls of other commercially available anti-Malware solutions, we employed machine learning techniques for realtime classification of such Adware on the device. This approach has the benefits of not requiring any sort of network connectivity to an update server, nor does it require connectivity for the actual detection stage. The system was tested on Adware, and, in addition, trained to classify Malware. As it turns out, a great proportion of Malware uses monetization schemes, such as intrusive ads and privacy invasion, that similar to Adware. Thanks to utilization of linear classifiers, it is possible to trace back the features responsible for detection. It is very instructive to see which features pop out in Malware and Adware. We believe that further analysis of the feature vector can lead to further insights on the behaviour of malicious software and of ways to filter the application markets from such software. It should be noted that that in real life, the majority of Android users download applications from official stores, for example Google Play or the Amazon appstore. These stores generally sweep their inventory and remove known Malware. Although zero day threats have been known to happen, they represent a negligible fraction of applications on the markets. Adware, on the other hand, do represent a significant portion of the markets and do pose a threat to user experience and users' privacy. By using Adware specific features, we are able to identify and consequently protect the users and their privacy.

Acknowledgment

The authors would like to thank the members of the research team in Shine Security for their help in building the Goodware, Malware and Adware databases, especially Itay Katz and Omri Ariav.

References

[1] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: mining API-level features for robust Malware detection in Android," *Security and Privacy in Communication Networks*, Springer Int. Publishing, 2013, pp. 86-103.

[2] D. Arp et al. "DREBIN: effective and explainable detection of Android Malware in your pocket," In *Proc. of the 21st Annu. Network and Distributed System Security Symp.* (NDSS), 2014.

[3] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273-297, 1995.

[4] H. Gascon et al. "Structural detection of Android Malware using embedded call graphs," In *Proc. of ACM CCS Workshop on Articial Intelligence and Security* (AISEC), 2013, pp. 45-54.

[5] M. Grace et al. "Riskranker: scalable and accurate zero-day Android Malware detection." In *Proc. of the 10th Int. Conf. on Mobile Systems, applications, and services*, ACM, 2012 pp. 281-294.

[6] M. Zheng, M. Sun, and J. Lui, "Droid Analytics: a signature based analytic system to collect, extract, analyze and associate Android Malware," In *Trust, Security and Privacy in Computing and Communication* (TrustCom), 12th IEEE Int. Conf., 2013, pp. 163-171.

[7] Y. Zhou et al. "Hey, You, Get Off of My Market: detecting malicious apps in official and alternative Android markets." In *Proc. of the 19th Annu. Network and Distributed System Security Symp.* (NDSS), San Diego, CA, 2012.

[8] Y. Zhou and X. Jiang, "Dissecting Android Malware: characterization and evolution," In *Security and Privacy* (SP), IEEE Symp., 2012, pp. 95-109.

[9] https://www.virustotal.com

[10] http://www.capitol-college.edu/