MADS: Malicious Android Applications Detection through String Analysis

Borja Sanz, Igor Santos, Javier Nieves, Carlos Laorden, Iñigo Alonso-Gonzalez, Pablo G. Bringas

S^{3}Lab, DeustoTech Computing, University of Deusto, Bilbao, Spain
`borja.sanz@deusto.es`, `isantos@deusto.es`, `jnieves@deusto.es`, `claorden@deusto.es`, `ialonso@opendeusto.es`, `pablo.garcia.bringas@deusto.es`

Abstract. The use of mobile phones has increased in our lives because they offer nearly the same functionality as a personal computer. Besides, the number of applications available for Android-based mobile devices has increased. Google offers to programmers the opportunity to upload and sell applications in the Android Market, but malware writers upload their malicious code there. In light of this background, we present here Malicious Android applications Detection through String analysis (MADS), a new method that extracts the contained strings from the Android applications to build machine-learning classifiers and detect malware.

Key words: malware, android, machine learning, security

1 Introduction

Smartphones have become very popular. They allow us to check the email, to browse the Internet, or to play games with our friends, wherever we are. But, in order to take advantage of every possibility these devices may offer, applications have to be previously installed in the devices.

In the past, the installation of applications was a source of problems for the users because there was not a centralized site for users to download their applications and they used to search them in the Internet. Several operating systems like Symbian, in an attempt to avoid piracy and protect the device, used an authentication protocol that certified the application and, usually, caused several inconveniences to the users (e.g., they could not install applications although they bought them).

Nowadays, new methods for the distribution and installation have appeared thanks to the widely used Internet connection in mobile devices. Therefore, users can install any application they want, avoiding the connection of the device to a personal computer. The App Store of Apple was the first online store to bring this new paradigm to novel users. The model was praised and it became very successful, leading to other vendors such as RIM, Microsoft or Google to adopt the same business model and developing application stores for their devices. These factors have led a large number of developers to focus on these platforms.

However, malware has also arrived to the application markets. To this end, both Android and iOS have different approaches to deal with malicious software. According to their response to the US Federal Communication Commission's July 2009[1], Apple applies a very strict review process by at least two reviewers. Android, on the other hand, relies on its security permission system and on the user's sound judgement. Unfortunately, users have usually no security consciousness and they do not read required permissions before installing an application.

Both AppStore and Android Market include in their terms of service, clauses that do not allow developers to upload malware to their markets but both markets have hosted malware. Therefore, we can conclude that both models by themselves are insufficient to ensure safety and other methods must be developed in order to enhance the security of the devices.

Machine learning classification has been widely used in malware detection [1–5]. Several approaches [6, 7] have been presented that focus on classifying executables specifying the malware category; e.g., Trojan horses, worms, viruses, or even the malware family.

Regarding Android, the number of new malware samples is also increasing exponentially and several approaches have already been proposed to detect malware. Shabtai et al. [8] built several machine learning models using as features: the count of elements, attributes and namespaces of the parsed Android Package File (.apk). To validate their models, they selected features using three selection methods: Information Gain, Fisher Score and Chi-Square. Their approach achieved 89% of accuracy classifying applications into only 2 categories: tools or games.

There are other proposals that use dynamic analysis for the detection of malicious applications. Crowdroid [9] is an approach that analyses the behaviour of the applications. Blasing et al. [10] created AASandbox, which is a hybrid dynamic-static approximation. The dynamic part is based on the analysis of the logs for the low-level interactions obtained during execution. Shabtai and Elovici [11] also proposed a Host-Based Intrusion Detection System (HIDS) which uses machine learning methods that determines whether the application is malware or not. Google has also deployed a framework for the supervision of applications called Bouncer. Oberheide and Miller [12] revealed how the system works: it is based in QEMU and it performs both static and dynamic analysis.

In light of this background, we present MADS (Malicious Android applications Detection through String analysis), a novel approach for detection of malware in Android. This method employs the strings contained in the disassembled Android applications, constructing a bag of words model in order to train machine-learning algorithms to provide detection of malicious applications.

In summary, our main contributions are:

[1] `http://online.wsj.com/public/resources/documents/wsj-2009-0731-FCCApple.pdf`

- We present a new technique for the representation of Android applications, based on the bag of words model formed by the strings contained in the disassembled application.
- We adapt well-known machine learning classifiers to provide detection of malicious applications in Android.
- We found out that machine-learning algorithms can provide detection of malicious applications in Android using the strings contained in the disassembled application as features.

The reminder of this paper is organized as follows. Section 2 presents and details MADS, our new approach to represent applications in order to detect malware in Android. Section 3 describes the machine-learning algorithms we have used. Section 4 describes the empirical evaluation of our method. Finally, section 5 discusses the obtained results and outlines the avenues of further work in this area.

2 Representation of Applications using String Analysis

One of the most widely-used techniques for classic malware detection is the usage of strings contained in the files [13, 2]. This technique extracts every character strings within an executable file. The information that may be found in these strings can be, for example, options in the menus of the application or malicious URLs to connect to. In this way, by means of an analysis of these data, it is possible to extract valuable information in order to determine whether an application is malicious or not.

The process that we followed in MADS is the following. We start by disassembling the application using the open-source Android disassembler `smali`[2]. Hereafter, we search for the `const-string` operation code within the disassembled code.

Using this disassembler, the representation of Android binaries are semantically richer than common desktop binaries. For example, the strings extraction in desktop binaries are complex and it is usual that malware writers obfuscate them to hide relevant information. Instead, the obfuscation of strings in the binaries of Android is more difficult, given the internal structure of the binaries in this platform.

In order to conform the strings, we tokenize the found symbols using the classic separators (e.g., dot, comma, colon, semi-colon, blank space, tab, etc.). In this way, we construct a text representation of an executable $\mathcal{E}$, that is formed by strings s_i, such as $\mathcal{E} = (s_1, s_2, ..., s_{n-1}, s_n)$ where n is the number of strings within a file.

$\mathcal{C}$ is the set of Android executables $\mathcal{E}$, $\{\mathcal{E} : \{s_1, s_2, ...s_n\}\}$, each comprising n strings $s_1, s_2, \ldots, s_n$, we define the weight $w_{i,j}$ as the number of times the string s_i appears in the executable $\mathcal{E}_j$ if s_i is not present in $\mathcal{E}$, $w_{i,j} = 0$. Therefore, an application $\mathcal{E}_j$ can be represented as the vector of weights $\boldsymbol{\mathcal{E}_j} = (w_{1,j}, w_{2,j}, ...w_{n,j})$.

[2] `http://code.google.com/p/smali/`

In order to represent string collection, a common approach in text mining area is to use the Vector Space Model (VSM) [14], which represents documents algebraically as vectors in a multidimensional space.

This space consists only of positive axis intercepts. Executables are represented by a string-by-executable matrix, where the $(i,j)^{th}$ element illustrates the association between the i^{th} string and the j^{th} executable. This association reflects the occurrence of the i^{th} string in executable j. Strings can represent can be individually weighted, allowing the strings to become more or less important within a given executable or the executable collection $\mathcal{C}$ as a whole.

We used the *Term Frequency – Inverse Document Frequency* (TF–IDF) [15] weighting schema, where the weight of the i^{th} string in the j^{th} executable, denoted by $weight(i,j)$, is defined by:

$$weight(i,j) = tf_{i,j} \cdot idf_i \tag{1}$$

where *term frequency* $tf_{i,j}$ is defined as:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2}$$

where $n_{i,j}$ is the number of times the string s_i appears in a executable $\mathcal{E}_j$, and $\sum_k n_{k,j}$ is the total number of strings in the executable $\mathcal{E}_j$. The inverse term frequency idf_i is defined as:

$$idf_i = \log\left(\frac{|\mathcal{C}|}{|\mathcal{C} : t_i \in \mathcal{E}|}\right) \tag{3}$$

where $|\mathcal{C}|$ is the total number of executables and $|\mathcal{C} : s_i \in \mathcal{E}|$ is the number of executables containing the string s_i.

Once we have characterized the application, we must classify it. In order to achieve it, we use machine learning algorithms. These algorithms allows us to, given a training dataset, assign a category (i.e., malware or goodware) to a evaluated sample.

3 Machine-learning Algorithms

Machine-learning is an active research area within *Artificial Intelligence* (AI) that focuses on the design and development of new algorithms that allow computers to reason and decide based on data [16].

Machine-learning algorithms can commonly be divided into three different types depending on the training data: supervised learning, unsupervised learning and semi-supervised learning. For supervised algorithms, the training dataset must be labelled (e.g., the class of an executable) [17]. Unsupervised learning algorithms try to determine how data are organised into different groups named clusters. Therefore, data do not need to be labelled [18]. Finally, semi-supervised machine-learning algorithms use a mixture of both labelled and unlabelled data in order to build models, improving the accuracy of solely unsupervised methods [19].

Because executables can be properly labelled, we use supervised machine-learning; however, in the future, we would also like to test unsupervised and semi-supervised methods for detection of malware in Android.

3.1 Bayesian Networks

Bayesian Networks [20], which are based on the *Bayes Theorem*, are defined as graphical probabilistic models for multivariate analysis. Specifically, they are directed acyclic graphs that have an associated probability distribution function [21]. Nodes within the directed graph represent problem variables (they can be either a premise or a conclusion) and the edges represent conditional dependencies between such variables. Moreover, the probability function illustrates the strength of these relationships in the graph [21].

The most important capability of Bayesian Networks is their ability to determine the probability that a certain hypothesis is true (e.g., the probability of an executable to be malware) given a historical dataset.

3.2 Decision Trees

Decision Tree classifiers are a type of machine-learning classifiers that are graphically represented as trees. Internal nodes represent conditions regarding the variables of a problem, whereas final nodes or leaves represent the ultimate decision of the algorithm [22].

Different training methods are typically used for learning the graph structure of these models from a labelled dataset. We use *Random Forest*, an ensemble (i.e., combination of weak classifiers) of different randomly-built decision trees [23], and *J48*, the WEKA [24] implementation of the *C4.5* algorithm [25].

3.3 K-Nearest Neighbour

The *K-Nearest Neighbour* (KNN) [26] classifier is one of the simplest supervised machine-learning models. This method classifies an unknown specimen based on the class of the instances closest to it in the training space by measuring the distance between the training instances and the unknown instance.

Even though several methods to choose the class of the unknown sample exist, the most common technique is to simply classify the unknown instance as the most common class amongst the K-nearest neighbours.

3.4 Support Vector Machines (SVM)

SVM algorithms divide the n-dimensional space representation of the data into two regions using a *hyperplane*. This hyperplane always maximises the *margin* between those two regions or classes. The margin is defined by the farthest distance between the examples of the two classes and computed based on the distance between the closest instances of both classes, which are called *supporting vectors* [27].

Instead of using linear hyperplanes, it is common to use the so-called *kernel functions*. These kernel functions lead to non-linear classification surfaces, such as polynomial, radial or sigmoid surfaces [28]

4 Experimental Results

In this section we describe the empirical validation of our method for Android malware applications detection.

4.1 Dataset Description

In this subsection, we detail how the dataset has been composed. The requirements that the final dataset has to meet are the following:

- It must be heterogeneous. It should show the diversity in the types of applications that are available in the Android market.
- It must be proportional to the number of samples that already exist of each type of application. To this end, two different datasets were created. The first one is composed by the benign applications whilst the second one is formed by malicious software.

Malicious Software To compile the malware dataset, the samples were obtained from the company VirusTotal[3]. VirusTotal offers a series of services called VirusTotal Malware Intelligence Services, which allow researchers to obtain samples from their databases.

To generate the dataset, we first selected the samples. Initially, we collected 2,808 samples. Next, we normalize the values given by the different antivirus vendors. The goal of this step was to determine their reliability detecting malware in Android.

To this end, we assumed that every sample that was detected as malware by at least one antivirus was malware, based on the given name of the antivirus. Then, we evaluated the detection rate of each antivirus engine with respect to the complete malware dataset:

$$a_w = \frac{n}{n_t} \tag{4}$$

where n is the number of samples detected by the antivirus and n_t is the total number of each antivirus detecting malware on the Android platform. Then, we evaluated each malware sample taking into account the weights of each antivirus.

For this evaluation, we applied the next metric:

$$m_w = \sum a_w | \forall a \in \mathcal{A} \tag{5}$$

[3] `http://www.virustotal.com`

being $\mathcal{A} = (a_1, a_2, , a_\ell)$ the set of the weights of the antivirus computed before, for the antiviruses that detect the sample. Therefore, m_w rates the detection taking into account the antiviruses that detect the sample.

We determined a threshold below which a sample cannot enter the dataset, in order to ensure that the samples belonging to it are relevant enough. The threshold was set empirically to 0.1, which provided us a total number of 1,202 malware samples. Besides, we focused on the results given by the different antiviruses to determine whether the samples were actually Android-based applications. This was performed using the naming convention of the different antivirus engines. Finally, we also removed any duplicated samples.

We finally acquired a malware dataset composed of 333 unique samples. According to the report elaborated by LookOut[4], this dataset represents the 75% of the malware that existed in July, 2011.

Benign Software To generate this dataset, we gathered 1,811 Android samples of diverse types. To classify them adequately, we categorised them using the same scheme that Android market follows. To this extent, we categorised the applications by means of an unofficial library called android-market-api[5]. Once the samples were classified, we selected a subgroup of samples to be part of the final benign software dataset. The employed methodology was the following:

Table 1. Number of samples for each category.

Arcade and Action	32	Multimedia & Video	23
Books	10	Music & Audio	12
Business	1	News & magazines	7
Card Games	2	Personalization	6
Casuals	10	Photography	6
Comics	1	Productivity	27
Communication	20	Puzzles	16
Education	0	Races	2
Enterprise	4	Sales	3
Entertainment	16	Society	25
Finance	3	Sports	5
Health	3	Tools	80
Libraries & Demos	2	Transportation	2
Lifestyle	4	Travels	8
Medicine	1	Weather	2

Total number of benign applications: 333

[4] `https://www.mylookout.com/_downloads/lookout-mobile-threat-report-2011.pdf`

[5] `http://code.google.com/p/android-market-api/`

1. *Determine the number of total samples.* To facilitate the training of the machine-learning models, it is usually desirable for both categories to be balanced. Therefore, given that the number of malware samples is inferior to the benign category, we opted to reduce the number of benign applications to 333.
2. *Determine the number of samples for each benign category.* Second, we decided to follow the proportion present in the Android market and, therefore, selected the number of applications accordingly.
3. *Types of application.* There are different types of applications: native ones (developed by means of the Android SDK), web (developed through HTML, JavaScript and CSS) and widgets (simple applications displayed in the Android desktop). All these applications have different features. To generate the dataset, we made no distinction in the type of application and included samples of the different types in the final dataset.
4. S*election of the samples for each category.* Once the number of applications for each category was determined, we selected the applications randomly using a Monte Carlo sampling method, avoiding different versions of the same application.

Following this methodology, we constructed the benign dataset. The number of samples for each category is shown in Table 1.

4.2 Configuration

For the evaluation of the different machine learning algorithms we used the tool WEKA (Waikato Environment for Knowledge Analysis) [24]. Specifically, the algorithms used in this tool can be seen in Table 2. In those cases in which no configuration parameters are specified, the configuration used was the default.

Table 2. Configuration of the algorithms.

Used Algorithms	**Configuration**
NaïveBayes	N/A
Bayessian Network	K2 and TAN
SVM	Polynomial and Normalized Polynomial Kernel
KNN	K: 1, 3 and 5
J48	N/A
RandomForest	N = 10, 50 and 100

The dataset was divided using the k-cross-validation technique [29, 30]. It divides k times the input dataset in k complementary subsets using one shaping sample data set, called test set, while the rest of subsets forming the joint training. To obtain the error ratio for the final sample, the arithmetic mean of the error rates obtained for each of the k iterations is calculated.

4.3 Evaluation

The evaluation was performed by measuring the following metrics:

- **True Positive Ratio (TPR).**

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

  where TP is the number of malware cases correctly classified (true positives) and FN is the number of malware cases misclassified as legitimate software (false negatives).
- **False Positive Ratio (FPR).**

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

  where FP is the number of benign software cases incorrectly detected as malware and TN is the number of legitimate executables correctly classified.
- **Accuracy.** It is the total number of the classifier's hits divided by the number of instances in the whole dataset:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{8}$$

- **Area under the ROC Curve (AUC).** AUC establishes the relation between false negatives and false positives [31]. The ROC curve is obtained by plotting the TPR against the FPR.

4.4 Results

Table 3 shows the obtained results for the tested algorithms.

Table 3. Obtained results.

Algorithm	*TPR*	*FPR*	*AUC*	*Accuracy (%)*
Naïve Bayes	0.93	0.17	0.90	88.07%
Bayesian Network: K2	0.71	0.13	0.89	78.68%
Bayesian Network: TAN	0.83	0.11	0.94	86,09%
SVM: Poly	0.93	0.03	0.95	94.70%
SVM: NPoly	0.77	0.04	0.86	86.45%
KNN K=1	0.35	0.02	0.84	66.24%
KNN K=3	0.22	0.03	0.82	59.85%
KNN K=5	0.17	0.03	0.79	56.75%
KNN K=10	0.08	0.04	0.77	51.77%
J48	0.83	0.12	0.86	85.54%
Random Forest N=10	0.92	0.13	0.96	89.74%
Random Forest N=50	0.93	0.09	0.97	91.81%
Random Forest N=100	0.94	0.09	0.97	92.04%

The best results were obtained the Random Forest configured with 100 trees, obtaining an AUC of 0.97 and an accuracy of 92.04%. Regarding TPR, this classifier can detect the 94% of the malware, whilst a 9% of the legitimate applications are misclassified. TPR and FPR establish the cost of misclassification. It is important to set the cost of false negatives $(1 - TPR)$ and false positives, in other words, establish whether is better to classify a malware as legitimate or to classify a benign software as malware. In particular, if our framework is devoted to detect new and unknown malware, one may think that it is more important to detect more malware than to minimise false positives. However, for commercial reasons, one may think just the opposite: a user can be bothered if their legitimate applications are flagged as malware. Therefore, we consider that the importance of the cost is established by the way our framework will be used. If it is used as a complement to standard anti-malware systems then we should focus on minimising false positives. Otherwise, if the framework is used within antivirus laboratories to decide which executables should be further analysed then we should minimise false negatives (or maximise true positives). To tune the proposed method, we can apply two techniques: (i) whitelisting and blacklisting or (ii) cost-sensitive learning. White and black lists store a signature of an executable in order to be flagged either as malware (blacklisting) or benign software (whitelisting). On the other hand, cost-sensitive learning is a machine-learning technique where one can specify the cost of each error and the classifiers are trained taking into account that consideration [32].

4.5 Comparison with Related Work

To combat the problem of malware that has risen in recent years in Android, researchers have begun to explore this area, using the experience acquired in other platforms.

"Andromaly"[33], a framework for detecting malware on Android mobile devices. This framework collected 88 features and events and, then, applied machine-learning algorithms to detect abnormal behaviours. Their dataset was composed of 4 self-written malware, as well as goodware samples, both separated into two different categories (games and tools). Their approach achieved a 0.99 area under ROC curve and 99% of accuracy. Despite these results, their framework had to collect a huge number of features and events, overloading the device and, consequently, draining the battery. Our approach only needs information extracted from .apk files, making the extraction process almost trivial. Although our results are not as sound as theirs, our approach requires less computational effort and our dataset is larger and sparser in malware samples than theirs.

On the other hand, Peng et al.[34] ranks the risks in Android using probabilistic generative models. They selected the permissions of the applications as key feature. Specifically, they chose the top 20 most frequently requested permissions in their dataset, composed by 2 benign software collections, obtained from the Google Play (157,856 and 324,658 samples, respectively) and 378 unique samples of malware. They obtained a 0.94 area under ROC curve as best result. We complemented the information provided by the permissions with the uses-features, enhancing the results and approaching them to those obtained by previous methods.

5 Discussion and Conclusions

Smartphones are a first class citizen nowadays. Unfortunately, malware writers are focused in this devices too. Malware detection techniques has moved from desktop to mobile devices. The main difference between both environment are available resources. Despite the evolution of the last years, current smartphones have several limitations (i.e., computational performance or battery life). Due to these limitations, the application of various techniques used in the desktop environment in smartphones is doubtful.

In this paper we propose a new method for detecting Android malware using string features to train machine-learning techniques. In order to validate our method, we collected several malware samples of Android applications. Then, we extracted the aforementioned features for each application and trained the models, evaluating each configuration. Random Forest was the best classifier obtaining very high accuracy levels. Nevertheless, there are several considerations regarding the viability of our approach.

The use of supervised machine-learning algorithms for the model training, can be a problem in itself. In our experiments, we used a training dataset that is very small when compared with commercial antivirus databases. As the dataset size grows, so does the issue of scalability. This problem produces excessive storage requirements, increases time complexity and impairs the general accuracy of the models [35]. To reduce disproportionate storage and time costs, it is necessary to reduce the original training set [36]. In order to solve this issue, *data reduction* is normally considered an appropriate preprocessing optimisation technique [37, 38]. Such techniques have many potential advantages such as reducing measurement, storage and transmission; decreasing training and testing times; confronting the *curse of dimensionality* to improve prediction performance in terms of speed, accuracy and simplicity and facilitating data visualization and understanding [39, 40]. Data reduction can be implemented in two ways. On the one hand, *Instance Selection* (IS) seeks to reduce the evidences (i.e., number of rows) in the training set by selecting the most relevant instances or re-sampling new ones [41]. On the other hand, *Feature Selection* (FS) decreases the number of attributes or features (i.e., columns) in the training set [42]. Both IS and FS are very effective at reducing the size of the training set and helping to filtrate and clean noisy data, thereby improving the accuracy of machine-learning classifiers [43, 44].

Besides, our method has several limitations due to the representation of executables. In this way, because the bag of words model is based on the frequencies with which strings appear within executables, malware writers may start modifying their techniques to evade filters. For example, in the field of spam filtering, *Good Word Attack* is a method that modifies the term statistics by appending a set of words that are characteristic of legitimate e-mails, thereby bypassing spam filters. In case that happens in our domain, we can adopt some of the methods that have been proposed, such as *Multiple Instance Learning* (MIL) [45]. MIL divides an instance or a vector in the traditional supervised learning methods into several sub-instances and classifies the original vector based on the sub-instances [46].

Morever, because of the static nature of the proposed method, it cannot counter *packed* malware. Packed malware is the result of cyphering the payload of the executable and deciphering it when the executable is finally loaded into memory. Indeed, static detection methods can deal with packed malware only by using the signatures of the packers. Accordingly, dynamic analysis seems a more promising solution to this problem [47]. Forensic experts are developing reverse engineering tools over Android applications, from which researchers could retrieve new features to enhance the data used to train the models.

Future work of this Android malware detection tool is oriented in three main directions. First, we will enhance the representation of data using data reduction techniques. Second, we will explore several attacks to this statistical model and propose solutions. Finally, we will use dynamically extracted features in order to improve our method.

Acknowledgements

This research was partially supported by the Basque Government under the research project 'BRANKA4U: Evolución de los servicios bancarios hacia el futuro' granted by the ETORGAI 2011 program.

References

1. Schultz, M., Eskin, E., Zadok, F., Stolfo, S.: Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001. S&P, IEEE (2001) 38–49
2. Santos, I., Devesa, J., Brezo, F., Nieves, J., Bringas, P.: Opem: A static-dynamic approach for machine-learning-based malware detection. In: International Joint Conference CISIS12-ICEUTE12-SOCO12 Special Sessions. Herrero, .; Snel, V.; Abraham, A.; Zelinka, I.; Baruque, B.; Quintin, H.; Calvo, J.L.; Sedano, J.; Corchado, E. (Eds.). Advances in Intelligent Systems and Computing. Volume 189. (2012) 97–108
3. Santos, I., Nieves, J., Bringas, P.G.: Semi-supervised learning for unknown malware detection. In: Proceedings of the 4^{th} International Symposium on Distributed Computing and Artificial Intelligence (DCAI). 9th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS). (2011) 415–422
4. Santos, I., Laorden, C., Bringas, P.G.: Collective classification for unknown malware detection. In: Proceedings of the 6^{th} International Conference on Security and Cryptography (SECRYPT). (2011) 251–256
5. Santos, I., Brezo, F., Ugarte-Pedrero, X., Bringas, P.G.: Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences **??**(??) ?? in press, DOI:10.1016/j.ins.2011.08.020.

6. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. Detection of Intrusions and Malware, and Vulnerability Assessment (2008) 108–125
7. Tian, R., Batten, L., Islam, R., Versteeg, S.: An automated classification system based on the strings of trojan and virus families. In: Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE). (2009) 23–30
8. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. In: Proceedings of the International Conference on Computational Intelligence and Security (CIS). (2010) 329–333
9. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM (2011) 15–26
10. Blasing, T., Batyuk, L., Schmidt, A., Camtepe, S., Albayrak, S.: An android application sandbox system for suspicious software detection. In: Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE). (2010) 55–62
11. Shabtai, A., Elovici, Y.: Applying behavioral detection on android-based devices. In: Proceedings of the 3rd International Conference on Mobile Wireless Middleware, Operating Systems, and Applications: Mobilware. Volume 48., Springer Verlag (2010) 235
12. Oberheide, J., Miller, J.: Dissecting the android bouncer. In: SUMERCON 2012. (2012) `http://jon.oberheide.org/files/summercon12-bouncer.pdf`.
13. Santos, I., Penya, Y., Devesa, J., Bringas, P.: N-Grams-based file signatures for malware detection. In: Proceedings of the 11^{th} International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS. (2009) 317–320
14. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
15. Salton, G., McGill, M.: Introduction to modern information retrieval. McGraw-Hill New York (1983)
16. Bishop, C.: Pattern recognition and machine learning. Springer New York. (2006)
17. Kotsiantis, S., Zaharakis, I., Pintelas, P.: Supervised machine learning: A review of classification techniques. Frontiers in Artificial Intelligence and Applications **160** (2007) 3
18. Kotsiantis, S., Pintelas, P.: Recent advances in clustering: A brief survey. WSEAS Transactions on Information Science and Applications **1**(1) (2004) 73–81
19. Chapelle, O., Schölkopf, B., Zien, A.: Semi-supervised learning. MIT Press (2006)
20. Pearl, J.: Reverend bayes on inference engines: a distributed hierarchical approach. In: Proceedings of the National Conference on Artificial Intelligence. (1982) 133–136
21. Castillo, E., Gutiérrez, J.M., Hadi, A.S.: Expert Systems and Probabilistic Network Models. Erste edn., New York, NY, USA (1996)
22. Quinlan, J.: Induction of decision trees. Machine learning **1**(1) (1986) 81–106
23. Breiman, L.: Random forests. Machine learning **45**(1) (2001) 5–32
24. Garner, S.: Weka: The Waikato environment for knowledge analysis. In: Proceedings of the 1995 New Zealand Computer Science Research Students Conference. (1995) 57–64
25. Quinlan, J.: C4. 5 programs for machine learning. Morgan Kaufmann Publishers (1993)

26. Fix, E., Hodges, J.L.: Discriminatory analysis: Nonparametric discrimination: Small sample performance. Technical Report Project 21-49-004, Report Number 11 (1952)
27. Vapnik, V.: The nature of statistical learning theory. Springer (2000)
28. Amari, S., Wu, S.: Improving support vector machine classifiers by modifying kernel functions. Neural Networks **12**(6) (1999) 783–789
29. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International joint Conference on artificial intelligence. Volume 14., LAWRENCE ERLBAUM ASSOCIATES LTD (1995) 1137–1145
30. Devijver, P., Kittler, J.: Pattern recognition: A statistical approach. Prentice/Hall International (1982)
31. Singh, Y., Kaur, A., Malhotra, R.: Comparative analysis of regression and machine learning methods for predicting fault proneness models. International Journal of Computer Applications in Technology **35**(2) (2009) 183–193
32. Elkan, C.: The foundations of cost-sensitive learning. In: Proceedings of the 2001 International Joint Conference on Artificial Intelligence. (2001) 973–978
33. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: andromaly: a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems (2012) 1–30
34. Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Using probabilistic generative models for ranking risks of android apps. In: Proceedings of the 2012 ACM conference on Computer and communications security, ACM (2012) 241–252
35. Cano, J., Herrera, F., Lozano, M.: On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining. Applied Soft Computing Journal **6**(3) (2006) 323–332
36. Czarnowski, I., Jedrzejowicz, P.: Instance reduction approach to machine learning and multi-database mining. In: Proceedings of the 2006 Scientific Session organized during XXI Fall Meeting of the Polish Information Processing Society, Informatica, ANNALES Universitatis Mariae Curie-Skłodowska, Lublin. (2006) 60–71
37. Pyle, D.: Data preparation for data mining. Morgan Kaufmann (1999)
38. Tsang, E., Yeung, D., Wang, X.: OFFSS: optimal fuzzy-valued feature subset selection. IEEE transactions on fuzzy systems **11**(2) (2003) 202–213
39. Torkkola, K.: Feature extraction by non parametric mutual information maximization. The Journal of Machine Learning Research **3** (2003) 1415–1438
40. Dash, M., Liu, H.: Consistency-based search in feature selection. Artificial Intelligence **151**(1-2) (2003) 155–176
41. Liu, H., Motoda, H.: Instance selection and construction for data mining. Kluwer Academic Pub (2001)
42. Liu, H., Motoda, H.: Computational methods of feature selection. Chapman & Hall/CRC (2008)
43. Blum, A., Langley, P.: Selection of relevant features and examples in machine learning. Artificial intelligence **97**(1-2) (1997) 245–271
44. Derrac, J., Garcıa, S., Herrera, F.: A First Study on the Use of Coevolutionary Algorithms for Instance and Feature Selection. In: Proceedings of the 2009 International Conference on Hybrid Artificial Intelligence Systems, Springer (2009) 557–564
45. Dietterich, T., Lathrop, R., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence **89**(1-2) (1997) 31–71
46. Maron, O., Lozano-Pérez, T.: A framework for multiple-instance learning. Advances in neural information processing systems (1998) 570–576

47. Kang, M., Poosankam, P., Yin, H.: Renovo: A hidden code extractor for packed executables. In: Proceedings of the 2007 ACM workshop on Recurring malcode. (2007) 46–53