Malware Detection with Confidence Guarantees on Android Devices

Nestoras Georgiou, Andreas Konstantinidis, Harris Papadopoulos

Malware Detection with Confidence Guarantees on Android Devices

Nestoras Georgiou, Andreas Konstantinidis, and Harris Papadopoulos*

Department of Computer Science and Engineering,
Frederick University, Cyprus

Abstract. The evolution of ubiquitous smartphone devices has given rise to great opportunities with respect to the development of applications and services, many of which rely on sensitive user information. This explosion on the demand of smartphone applications has made them attractive to cybercriminals that develop mobile malware to gain access to sensitive data stored on smartphone devices. Traditional mobile malware detection approaches that can be roughly classified to signature-based and heuristic-based have essential drawbacks. The former rely on existing malware signatures and therefore cannot detect zero-day malware and the latter are prone to false positive detections. In this paper, we propose a heuristic-based approach that quantifies the uncertainty involved in each malware detection. In particular, our approach is based on a novel machine learning framework, called Conformal Prediction, for providing valid measures of confidence for each individual prediction, combined with a Multilayer Perceptron. Our experimental results on a real Android device demonstrate the empirical validity and both the informational and computational efficiency of our approach.

Keywords: Malware Detection; Android; Security; Inductive Conformal Prediction; Confidence Measures; Multilayer Perceptron

1 Introduction

The widespread deployment of smartphone devices has brought a revolution in mobile applications and services that span from simple messaging and calling applications to more sensitive financial transactions and internet banking services. As a result, a great deal of sensitive information, such as access passwords and credit card numbers, are stored on smartphone devices, which has made them a very attractive target for cybercriminals. More specifically, a significant increase of malware attacks was observed in the past few years, aiming at stealing private information and sending it to unauthorized third-parties.

Mobile malware are malicious software used to gather information and/or gain access to mobile computer devices such as smartphones or tablets. In particular, they are packaged and redistributed with third-party applications to

* Corresponding author. Email: h.papadopoulos@frederick.ac.cy

inject malicious content into a smartphone and therefore expose the device's security. While the first one appeared in 2004 targeting the Nokia Symbian OS [1], in the fourth quarter of 2015 G DATA security experts reported discovering 8,240 new malware applications on average per day and a total of 2.3 million new malware samples in 2015, in just the Android OS [2]. When malware compromises a smartphone, it can illegally watch and impersonate its user, participate in dangerous botnet activities without the user's consent and capture user's personal data.

Mobile malware detection techniques can be classified into two major categories: static analysis [4, 20] and dynamic analysis [7, 18, 14]. The former aim at detecting suspicious patterns by inspecting the source code or binaries of applications. However, malware developers bypass static analysis by employing various obfuscation techniques and therefore limiting their ability to detect polymorphic malware, which change form in each instance of the malware [13]. Dynamic analysis techniques on the other hand, involve running the application and analyzing its execution for suspicious behavior, such as system calls, network access as well as file and memory modifications. The main drawback of these techniques is that it is difficult to determine when and under what conditions the malware malicious code will be executed.

Both static and dynamic analysis techniques are typically implemented following two main approaches: signature-based approaches, which identify known malware based on unique signatures [9], and heuristic based approaches, which identify malicious malware behaviour based on rules developed by experts or by employing machine learning techniques [8, 20, 14, 12]. Even though signature-based techniques have been successfully adopted by antivirus companies for malware detection in desktop applications, this is not a preferred solution in the case of mobile devices due to their limited available resources in terms of power and memory. Additionally, signature-based techniques cannot detect zero-day malware (not yet identified) or polymorphic malware, something that is not an issue for heuristic based techniques. On the other hand, unlike signature-based techniques, heuristic based techniques are prone to false positive detections (i.e. wrongly identifying an application as malware).

Most recent research studies focus on extending the idea of heuristic-based approaches by employing machine learning techniques. For example, Sahs and Khan [19] use a one-class Support Vector Machine to detect malicious applications based on features extracted from Android Application Packages (APKs) of benign applications only. Demertzis and Iliadis [6] propose a hybrid method that combines Extreme Learning Machines with Evolving Spiking Neural Networks using features extracted from the behaviour of applications when executed on an emulated Android environment. In [5] the same authors propose an extension to the Android Run Time Virtual Machine architecture that analyses the Java classes of applications using the Biogeography-Based Optimizer (BBO) heuristic algorithm for training a Multilayer Perceptron to classify applications as malicious or benign. Abah et. al. [11] present a detection system that uses a k-Nearest Neighbour classifier to detect malicious applications based on features extracted

during execution. To the best of our knowledge, none of the machine learning based methods proposed in the literature provides any reliable indication on the likelihood of its detections being correct.

This paper proposes an approach that addresses the uncertainty arising from the use of machine learning techniques. Since there is no way of guaranteeing 100% accuracy with any machine learning technique, this study aims at providing a reliable indication of how probable it is for a prediction to be correct. The provision of such an indication for applications identified as possible malware, would be of great value for the decision of the user on whether to remove an application or not, depending on the risk he/she is willing to take. The proposed approach utilizes a novel machine learning framework, called Conformal Prediction (CP) [21], for providing provably valid measures of confidence for each individual prediction without assuming anything more than that the data are generated independently from the same probability distribution (i.i.d.). The confidence measures produced by CP have a clear probabilistic interpretation, thus enabling informed decision making based on the likelihood of each malware detection being correct.

Specifically, we combine the inductive version of the CP framework [17, 16], which is much more computationally efficient than the original version, with a Multilayer Perceptron (MLP), which is one of the most popular and well-performing machine learning techniques. We evaluated the proposed approach on a realistic malware dataset using a prototype developed for a real Android device (LG E400). Our experimental results demonstrate the empirical validity of our approach as well as its efficiency in terms of accuracy, informativeness and computational time.

The rest of the paper starts with an overview of the Conformal Prediction framework and its inductive counterpart in Section 2. The next section (Section 3) details the proposed approach. Section 4 describes the dataset used for evaluating our approach, while Section 5 presents our experiments and the obtained results. Finally, Section 6 gives our conclusions.

2 Conformal Prediction

The Conformal Prediction (CP) framework extends conventional machine learning algorithms into techniques that produce reliable confidence measures with each of their predictions. The typical classification task consists of a training set $\{(x_1, y_1), \ldots, (x_l, y_l)\}$ of instances $x_i \in \mathbb{R}^d$ together with their associated classifications $y_i \in \{Y_1, \ldots, Y_c\}$ and a new unclassified instance x_{l+1}. The aim of Conformal Prediction is not only to find the most likely classification for the unclassified instance, but to also state something about its confidence in each possible classification.

CP does this by assigning each possible classification $Y_j, j = 1, \ldots, c$ to x_{l+1} in turn and extending the training set with it, generating the set

$$\{(x_1, y_1), \ldots, (x_l, y_l), (x_{l+1}, Y_j)\}. \quad (1)$$

It then measures how strange, or non-conforming, each pair (x_i, y_i) in (1) is for the rest of the examples in the same set. This is done with a *non-conformity measure* which is based on a conventional machine learning algorithm, called the *underlying algorithm* of the CP. This measure assigns a numerical score α_i to each pair (x_i, y_i) indicating how much it disagrees with all other pairs in (1). In effect it measures the degree of disagreement between the prediction of the underlying algorithm for x_i after being trained on (1) with its actual label y_i; in the case of x_{l+1}, y_{l+1} is assumed to be Y_j.

To convert the non-conformity score $\alpha_{l+1}^{(Y_j)}$ of (x_{l+1}, Y_j) into something informative, CP compares it with all the other non-conformity scores $\alpha_i^{(Y_j)}, i = 1, \ldots, l$. This comparison is performed with the function

$$p((x_1, y_1), \ldots, (x_l, y_l), (x_{l+1}, Y_j)) = \frac{|\{i = 1, \ldots, l+1 : \alpha_i^{(Y_j)} \geq \alpha_{l+1}^{(Y_j)}\}|}{l+1}. \quad (2)$$

The output of this function, which lies between $\frac{1}{l+1}$ and 1, is called the p-value of Y_j, also denoted as $p(Y_j)$, as this is the only unknown part of (1). If the data are independent and identically distributed (i.i.d.), the output $p(y_{l+1})$ for the true classification of x_{l+1} has the property that $\forall \delta \in [0, 1]$ and for all probability distributions P on Z,

$$P^{l+1}\{((x_1, y_1), \ldots, (x_{l+1}, y_{l+1})) : p(y_{l+1}) \leq \delta\} \leq \delta; \quad (3)$$

for a proof see [15]. Therefore all classifications with a p-value under some very low threshold, say 0.05, are highly unlikely to be correct as such sets will only be generated at most 5% of the time by any i.i.d. process.

Based on the property (3), given a *significance level* δ, or confidence level $1 - \delta$, a CP calculates the p-value of all possible classifications Y_j and outputs the prediction set

$$\{Y_j : p(Y_j) > \delta\}, \quad (4)$$

which has at most δ chance of not containing the true classification of the new unclassified example. In the case where a single prediction is desired (forced prediction) instead of a prediction set, CP predicts the classification with the largest p-value, which is the most likely classification, together with a confidence and a credibility measure for its prediction. The confidence measure is calculated as one minus the second largest p-value, i.e. the significance level at which all but one classifications would have been excluded. This gives an indication of how likely the predicted classification is compared to all other classifications. The credibility measure on the other hand, is the p-value of the predicted classification. A very low credibility measure indicates that the particular instance seems very strange for all possible classifications.

2.1 Inductive Conformal Prediction

The transductive nature of the original CP technique, which means that all computations have to start from scratch for every new test example, makes it

too computationally demanding for a mobile phone application. For this reason the proposed approach follows the inductive version of the framework, called Inductive Conformal Prediction (ICP), which only performs one training phase to generate a general rule with which it can then classify new examples with minimal processing.

Specifically, ICP divides the training set (of size l) into the *proper training set* with $m < l$ examples and the *calibration set* with $q := l - m$ examples. It then uses the proper training set for training the underlying algorithm (only once) and the examples in the calibration set for calculating the p-value of each possible classification of the new test example. In effect, after training the underlying algorithm on the proper training set the non-conformity scores $\alpha_{m+1}, \ldots, \alpha_{m+q}$ of the calibration set examples are calculated. Then to calculate the p-value of each possible classification $Y_j \in \{Y_1, \ldots, Y_c\}$ of a new test example x_{l+1}, ICP only needs to calculate the non-conformity score of the pair (x_{l+1}, Y_j) using the already trained underlying algorithm and compare it to the non-conformity scores of the calibration set examples with the function

$$p(Y_j) = \frac{|\{i = m+1, \ldots, m+q, l+1 : \alpha_i \geq \alpha_{l+1}^{(Y_j)}\}|}{q+1}. \tag{5}$$

Notice that the steps that need to be repeated for each test example have almost negligible computational requirements.

3 Proposed Approach

In this study a Multilayer Perceptron (MLP) was used as underlying algorithm of the ICP. The MLP used was a 2-layer fully connected feed-forward network with a sigmoid activation function in all units. It was implemented using the Multilayer Perceptron class of the WEKA data mining software libraries [10]. When given an instance x_i to classify, the trained MLP produces a probabilistic value $\hat{P}(Y_j|x_i)$ for each classification Y_j. In this work the classification task is binary, therefore $Y_j \in \{0, 1\}$ and two probabilistic values are produced by the MLP: $\hat{P}(0|x_i)$ and $\hat{P}(1|x_i)$.

The nonconformity measure used for the proposed MLP-ICP was

$$\alpha_i^{Y_j} = 1 - \hat{P}(y_i|x_{m+i}), \quad i = 1, \ldots, q, \tag{6a}$$

$$\alpha_{l+1}^{Y_j} = 1 - \hat{P}(Y_j|x_{l+1}), \tag{6b}$$

where $\hat{P}(y_i|x_i)$ is the output of the MLP for the true classification of x_i and $\hat{P}(Y_j|x_{l+1})$ is the output of the MLP for the assumed class $Y_j \in \{0, 1\}$ of x_{l+1}.

The complete process followed by the proposed approach is detailed in Algorithm 1. Lines 1 to 5 correspond to the training phase that needs to be performed only once. This phase trains the MLP on the proper training set (line 1) and calculates the nonconformity score of each calibration example x_{m+i}, $i = 1, \ldots, q$, by inputing it to the trained MLP to obtain the probabilistic outputs $\hat{P}(0|x_{m+i})$

Algorithm 1: Binary MLP-ICP

Input: proper training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$,
calibration set $\{(x_{m+1}, y_{m+1}), \ldots, (x_{m+q}, y_{m+q})\}$,
test example x_{l+1},
significance level δ

1 $h \leftarrow$ train the MLP on $\{(x_1, y_1), \ldots, (x_m, y_m)\}$;
2 **for** $i = 1$ **to** q **do**
3 $\quad \{\hat{P}(0|x_{m+i}), \hat{P}(1|x_{m+i})\} \leftarrow h(x_{m+i})$;
4 $\quad \alpha_{m+i} \leftarrow 1 - \hat{P}(y_{m+i}|x_{m+i})$;
5 **end**
6 $\{\hat{P}(0|x_{l+1}), \hat{P}(1|x_{l+1})\} \leftarrow h(x_{l+1})$;
7 **for** $j = 0$ **to** 1 **do**
8 $\quad \alpha_{l+1}^{j} \leftarrow 1 - \hat{P}(j|x_{l+1})$;
9 $\quad p(j) = \frac{|\{i=1,\ldots,q : \alpha_{m+i} \geq \alpha_{l+1}^{j}\}|+1}{q+1}$;
10 **end**

Output:

11 Prediction set $R \leftarrow \left\{Y_j : p(Y_j) > \delta\right\} \cup \left\{\arg\max_{k=1,\ldots,c} p(Y_k)\right\}$.

and $\hat{P}(1|x_{m+i})$ (line 3) and using them in (6a) to calculate α_{m+i} (line 4). The testing phase, lines 6 to 11, is the only part that needs to be repeated for every new instance. This phase obtains the probabilistic outputs of the trained MLP for the new instance (line 6) and then for each possible class, it calculates the corresponding nonconformity score with (6b) in line 8 and uses it together with the nonconformity scores of the calibration examples in (5) to calculate the p-value of the new instance belonging to that class (line 9). Finally, in line 11, it outputs the prediction set (4).

4 Data Used

For evaluating the proposed approach a malware dataset[1] created by B. Amos [3] was used. The data was collected using a shell script to automatically record the behavior of Android application files (*.apk*) by installing and running them on an Android emulator. After each installation the emulator simulated random user interaction with the application using the Android "*adb-monkey*" tool. The data collected for each application included Binder, Battery, Memory, CPU, Network, and Permission information. The battery related features were removed in our experiments since they all had the same value across all instances, as they were collected through an emulator. Table 1 presents all the features used in our experiments.

The dataset consisted of 6832 data samples in total, with 5121 benign samples and 1711 malicious samples described by 40 features (after the removal of

[1] Available on-line at: https://github.com/VT-Magnum-Research/antimalware

Table 1. Features used in this work divided into categories.

Category	Features
Binder	Transaction, Reply, Acquire, Release, ActiveNodes, TotalNodes, ActiveRef, TotalRef, ActiveDeath, TotalDeath, ActiveTransaction, TotalTransaction, ActiveTransactionComplete, TotalTransactionComplete, TotalNodesDiff, TotalRefDiff, TotalDeathDiff, TotalTransactionDiff, TotalTransactionCompleteDiff
CPU	User, System, Idle, Other
Memory	Active, Inactive, Mapped, FreePages, AnonPages, FilePages, DirtyPages, WritebackPages
Network	TotalTXPackets, TotalTXBytes, TotalRXPackets, TotalRXBytes, TotalTXPacketsDiff, TotalTXBytesDiff, TotalRXPacketsDiff, TotalRXBytesDiff
Permissions	TotalPermissions

the battery related data). In our experiments the data samples were divided randomly into a training set consisting of 6165 samples (4617 benign and 1548 malicious) and a test set consisting of 667 samples (504 benign and 163 malicious). For MLP-ICP the training set was further divided (also randomly) into the proper training set with 5466 samples (4097 benign and 1369 malicious) and the calibration set with 699 samples (520 benign and 179 malicious). All experiments were repeated 10 times with different random divisions of the data so as to ensure that the obtained results are not dependent on a particular division.

5 Experimental Results

In order to evaluate the proposed approach in the environment it is intended for, both the MLP-ICP and its underlying algorithm were implemented on the Android OS using the WEKA data mining software libraries and all experiments were performed on a LG E400 Android device. Before each experiment all input features were normalized to the range [0,1], based only on the training set of each of the 10 random divisions of the data and the resulting transformation was applied to the test set.

5.1 Accuracy

Our first set of experiments evaluates the proposed technique in terms of the accuracy of forced predictions. That is when the MLP-ICP predicts the most likely classification together with a confidence and credibility measure to that classification. We compare the accuracy of the proposed approach to that of its underlying algorithm. Note that the aim of the proposed approach is not to

Table 2. Forced prediction performance of the proposed approach compared to its underlying algorithm. The standard deviation of each value is given in the brackets.

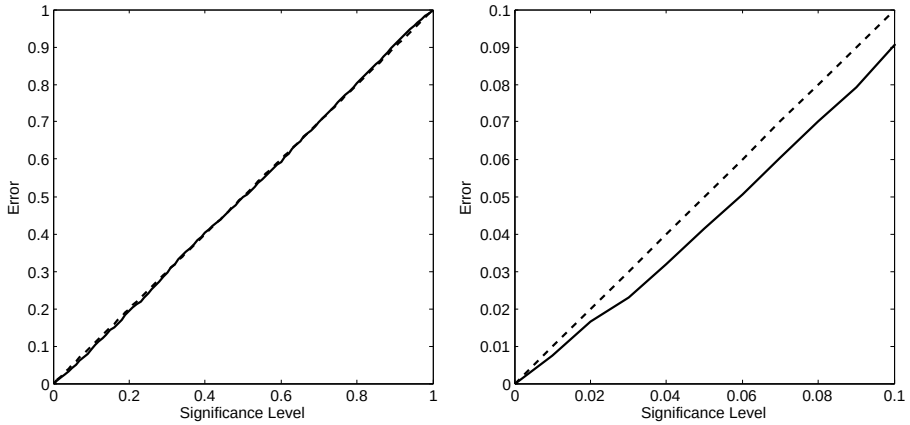
	Accuracy (%)	Mean Confidence (%)
MLP-ICP	94.24 (0.0116)	98.83 (0.0020)
Conventional MLP	94.17 (0.0117)	-

improve performance in terms of accuracy, but rather to provide probabilistically valid additional information about the likelihood of each prediction being correct, which will aid user decision making. Therefore our aim here is to examine if any degree of accuracy is sacrificed by the proposed approach in order to provide this additional information.

Table 2 reports the mean accuracy of the proposed approach and of the conventional MLP over 10 experimental runs with different random divisions of the data. For MLP-ICP we also report the mean of the confidence values produced. The values in brackets are the standard deviation of the corresponding values over the 10 experiments. The reported results show that there is no significant difference in terms of the accuracy of the two approaches; in fact the MLP-ICP gives higher accuracy, but the difference is insignificant. This shows that there is no "price to pay" in terms of accuracy for obtaining the much more informative outputs of the proposed approach. Furthermore, the high mean confidence produced by the proposed approach gives a first indication of the usefulness of its outputs. Finally, the small standard deviation of both accuracy and mean confidence over the 10 experimental runs shows that the presented mean values are a reliable indication of performance.

5.2 Empirical Validity

Next we evaluate the empirical validity of the prediction sets, and indirectly of the p-values and confidence measures, produced by the proposed approach. Figure 1 plots the error percentages of these prediction sets as the significance level changes. The left pane displays the error percentages for all significance levels $\delta \in [0, 1]$, while the right pane displays the lower left corner of the first plot, for $\delta \in [0, 0.1]$, which is the most important part of the plot as we are generally interested in high confidence levels. The solid line represents the actual error percentages observed, while the dashed line represents the diagonal, i.e. where the error is equal to the required significance level δ. The closeness of the actual errors to the diagonal confirms the empirical validity of the proposed approach, i.e. the errors made are always extremely near the required significance level; the very small deviation of about 0.01 visible in the left pane is due to statistical fluctuations. This confirms empirically the guaranteed validity of the ICP.



(a) $\delta \in [0, 1]$ (b) $\delta \in [0, 0.1]$

Fig. 1. Empirical validity of the proposed approach. The error percentage of the ICP is plotted with a solid line and the diagonal (exact validity) with a dashed line.

5.3 Quality of p-values

To evaluate the quality of the resulting p-values we examine the percentage of prediction sets that contained both classifications at different confidence levels. This is the percentage of instances for which the ICP was not able to reject one of the two possible classes at the required confidence level, we call these *uncertain prediction sets*. The lower these values are the more the instances for which the prediction sets are informative at the given confidence level and the higher the quality of the p-values used to compute them.

Figure 2 plots the percentage of uncertain prediction sets for the range of significance levels $\delta \in [0, 0.1]$, i.e. confidence levels between 100% and 90%. The plot shows that the prediction sets produced by the proposed approach are quite informative. With a confidence level as high as 99%, i.e. guaranteeing only 1% of errors, we can be certain in 63.7% of applications on whether they are malware or not. If we decrease the required confidence level by 1% (to 98%) we obtain certain predictions for 80.28% of the cases. Further lowering the confidence level to 95%, which is still rather high, increases the percentage of certain predictions to 95.53%. While for confidence levels 93% and 92% the number of certain predictions rises to more than 99% and becomes 100% from 91% confidence and below. This means that we can have a certain prediction for the large majority of applications with a very small risk, while for applications with uncertain predictions further investigation might be the best option.

The informativeness of the obtained prediction sets also shows the quality of the p-values used for constructing them, and as a result of the confidence measures produced based on the same p-values in the forced prediction case.

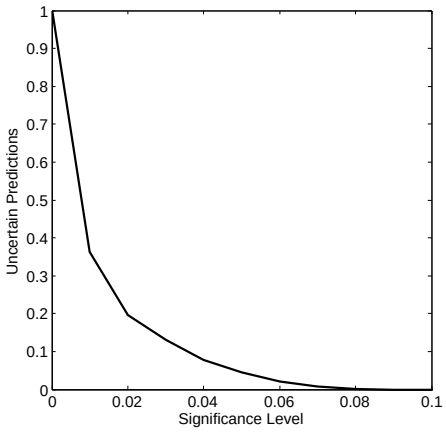


Fig. 2. Percentage of uncertain prediction sets for significance levels $\delta \in [0, 0.1]$.

In this case, the user can decide based on the confidence measure for a given malware prediction and on the risk he/she is willing to take on whether to remove it or not.

5.4 Computational Efficiency

Finally, since in a mobile device environment the amount of recourses needed are an important consideration, we examine the execution time of the proposed approach on a LG E400 Android device and compare it to that of its underlying algorithm. The training time for the MLP-ICP was 21 minutes, while the training time for the conventional MLP was 29 minutes. The reason for the lower training time of the ICP is the removal of the calibration examples from the proper training set. This works in favour of the proposed approach in terms of computational efficiency, while it has no negative impact on its accuracy as shown in Subsection 5.1. Although the training time needed seems somewhat long, it should be kept in mind that training only needs to be performed once and this can even be done on a desktop device. The testing time for both methods was 0.005 seconds, which is very computationally efficient for a mobile device.

6 Conclusions

This work proposed a machine learning approach for Android malware detection, which unlike traditional machine learning based malware detection techniques, produces valid confidence measures in each of its predictions. Such measures enable the user to take informed decisions on whether to remove an application identified as a possible malware, knowing the risk associated with each decision. The proposed approach is based on the Conformal Prediction framework, which

produces provably valid confidence measures that have a clear probabilistic interpretation without assuming anything more than i.i.d. data.

Our experimental results on a real Android device have shown that the proposed approach gives high accuracy, which is equal to (or even better than) that of the conventional machine learning approach it is based on. Furthermore, they demonstrate the empirical validity and high informational efficiency of the produced prediction sets and confidence measures. Finally in terms of computational efficiency, the training time needed by our approach is smaller than that of the conventional MLP on which it bases its predictions. While the time needed for classifying a new instance is adequately small for the limited resources of mobile devices.

The particular CP used in this work is based on MLP, which is one of the most popular machine learning techniques. However, the same general approach proposed here can be followed with other high performing machine learning techniques preserving the desirable properties observed in our experiments. The evaluation and performance comparison of CPs based on different conventional machine learning techniques for the particular task is our immediate future plan. Additionally the collection of a dataset from actual mobile devices rather than in an emulated environment in order to study the performance of the proposed approach on more realistic data is another future goal.

References

1. Cabir, smartphone malware. http://www.f-secure.com/v-descs/cabir.shtml (2004 - accessed May 12, 2016)
2. G DATA, mobile malware report (threat report: Q4/2015). https://secure.gd/dl-us-mmwr201504 (2016 - accessed May 16, 2016)
3. Amos, B., Turner, H., White, J.: Applying machine learning classifiers to dynamic android malware detection at scale. In: Proceedings of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013). pp. 1666–1671. IEEE (2013)
4. Christodorescu, M., Jha, S.: Static analysis of executables to detect malicious patterns. In: Proceedings of the 12th conference on USENIX Security Symposium. vol. 12, pp. 12–12. USENIX Association (2003)
5. Demertzis, K., Iliadis, L.: Same: An intelligent anti-malware extension for android art virtual machine. In: Núñez, M., Nguyen, T.N., Camacho, D., Trawiński, B. (eds.) Computational Collective Intelligence: 7th International Conference (ICCCI 2015), Part II. LNCS, vol. 9330, pp. 235–245. Springer (2015)
6. Demertzis, K., Iliadis, L.: Bio-inspired hybrid intelligent method for detecting android malware. In: Kunifuji, S., Papadopoulos, A.G., Skulimowski, M.A., Kacprzyk, J. (eds.) Knowledge, Information and Creativity Support Systems: Selected Papers from KICSS'2014. pp. 289–304. Springer (2016)
7. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys 44(2), 6:1–6:42 (2012), http://doi.acm.org/10.1145/2089125.2089126
8. Grégoire Jacob, Hervé Debar, E.F.: Behavioral detection of malware: from a survey towards an established taxonomy. Journal in Computer Virology 4(3), 251–266 (2008)

9. Griffin, K., Schneider, S., Hu, X., Chiueh, T.C.: Automatic generation of string signatures for malware detection. In: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. pp. 101–120. LNCS, Springer (2009)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations Newsletter 11(1), 10–18 (2009), http://doi.acm.org/10.1145/1656274.1656278
11. Joshua, A., Waziri, O.V., Abdullahi, M.B., Arthur, U.M., Adewale, O.S.: A machine learning approach to anomaly-based detection on android platforms. International Journal of Network Security and Its Applications 7(6), 15–35 (2015)
12. Menahem, E., Shabtai, A., Rokach, L., Elovici, Y.: Improving malware detection by applying multi-inducer ensemble. Computational Statistics and Data Analysis 53(4), 1483–1494 (2009)
13. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference. pp. 421–430. IEEE (2007)
14. Moskovitch, R., Elovici, Y., Rokach, L.: Detection of unknown computer worms based on behavioral classification of the host. Computational Statistics and Data Analysis 52(9), 4544–4566 (2008)
15. Nouretdinov, I., Vovk, V., Vyugin, M.V., Gammerman, A.: Pattern recognition and density estimation under the general i.i.d. assumption. In: Proceedings of the 14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory. Lecture Notes in Computer Science, vol. 2111, pp. 337–353. Springer (2001)
16. Papadopoulos, H.: Inductive Conformal Prediction: Theory and application to neural networks. In: Fritzsche, P. (ed.) Tools in Artificial Intelligence, chap. 18, pp. 315–330. InTech, Vienna, Austria (2008), http://www.intechopen.com/download/pdf/pdfs_id/5294
17. Papadopoulos, H., Proedrou, K., Vovk, V., Gammerman, A.: Inductive confidence machines for regression. In: Proceedings of the 13th European Conference on Machine Learning (ECML'02). LNCS, vol. 2430, pp. 345–356. Springer (2002)
18. Rieck, K., Holz, T., Willems, C., Düssel, P., Laskov, P.: Learning and classification of malware behavior. In: Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and vulnerability Assessment. LNCS, vol. 5137, pp. 108–125. Springer (2008)
19. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: Proceedings of the 2012 European Intelligence and Security Informatics Conference (EISIC). pp. 141–147. IEEE (2012)
20. Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C.: Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Information Security Technical Report 14(1), 16–29 (2009)
21. Vovk, V., Gammerman, A., Shafer, G.: Algorithmic Learning in a Random World. Springer, New York (2005)