Yes, Machine Learning Can Be More Secure! A case study on Android Malware Detection.

Sarat Sai Bammidi, Masters in Information Systems Security Concordia University Montreal, Canada.

E-mail: sai.sarat2@gmail.com

***Abstract*— Machine learning has been widely embraced as a statistically sound malware detection technology to handle the rising variety and sophistication of modern threats. Yet, new research has not only called into question how well-designed attacks will fare against it, but it has also demonstrated that machine learning contains built-in flaws that can be used to avoid detection during testing. In Alternatively stated, machine learning itself may be the weakest link in a security system. In this study, we model attackers with varying skills and capacities to categorize probable attack scenarios against learning-based malware detection techniques. We base our classification on an attack architecture that has been previously described. To fully evaluate the security of Drebin, an Android malware detector, we then construct and put into practice a set of corresponding evasion attacks. This study's key contribution is the suggestion of a straightforward and adaptable secure-learning paradigm that lessens the effects of evasion assaults while only marginally decreasing the detection rate in the absence of an attack. Finally, we contend that other malware detection jobs can also benefit from the easy use of our secure-learning methodology.**

***Keywords*- Android malware detection, static analysis, secure machine learning, computer security, Debrin.**

1 INTRODUCTION

As the days passing the machine learning plays an important role in security, as training and test data are assumed to follow the same underlying probability distribution, machine learning algorithms are susceptible to well-planned attacks that violate this assumption. This implies that the security chain's weakest link may be machine learning itself [1]. Attackers with the necessary skills can alter data during testing to avoid detection or introduce poison samples into training data to trick the learning algorithm and result in misclassification mistakes.[2][3][4][5][6][7]

The machine learning method is built from the ground up to be more resistant to evasion in this article using the *adversary strategy*. We use Android malware detection as a case study for our methodology. Android has over a billion users worldwide, making it the most widely used mobile operating system. At the same time, the number of malicious programs targeting it has increased, demonstrating the importance of this task.

Here, we concentrate our investigation on Drebin, a machine-learning method that uses static analysis to effectively detect Android malware on mobile devices [8]. The focus of our analysis is on debrin, which is a machine learning approach which relies on the static analysis for the effective direct mobile device detection of Android malware. In the paper our aim is to improve the security of Drebin against the stealthier attacks as the static analysis leaves some detectable traces. In order to perform a well security analysis we exploit an adversarial framework based on the previous work on the adversarial machine learning. Here we use the commercial tool DexGuard, it was initially created to make it more challenging to reverse-engineering benign applications.

The secure machine learning algorithm used is completely new, with respect to the previous techniques for secure learning [9][10]. It can maintain scalability and computational effectiveness on. It is well-motivated from a more theoretical perspective and can handle big datasets (since it uses a linear classification function) perspective. We use real-world data to empirically test our method.

The key drawbacks of our research are finally covered, along with potential future research directions and how to use the suggested methodology for various malware detection jobs.

2 Android Malware Detection

We provide some background information on Android applications in this area. The main drawbacks of Drebin are then discussed.

2.1 Android Background

All the android applications are built in the APK format which is a zip file and contains two files android manifest and classes.dex.

Android Manifest: The Android operating system's Android Manifest is an XML file that contains crucial details about an Android application. Every Android application must include this file, which is essential since it provides vital details like the package name, application name, version code, version name, permissions, activities, services, receivers, and more. The manifest also lists the hardware components and permissions that the application needs in order to function.

Dalvik Bytecode (dexcode): An application's-built source code can be found in the classes.dex file. All the user-implemented classes and methods are contained there. There may be special API calls in Classes.dex that can access private resources like contacts that are sensitive (suspicious calls). It also includes all system-related, restricted API calls whose functionality needs permissions (e.g., using the Internet).

Finally, this file may have references to network addresses that the application may contact.

2.2 Drebin

Android malware is found and categorized by the Drebin system using a combination of static and dynamic analysis approaches. Although the dynamic analysis entails executing

the Android application package (APK) in a sandboxed environment to track its behavior, the static analysis involves looking through the APK file for questionable coding patterns and features.

TABLE 1
Overview of Feature sets

	Feature sets	
Manifest	S1	Hardware Components
	S2	Requested Permissions
	S3	Application Components
	S4	Filtered intents
Dexcode	S5	Restricted API calls
	S6	Used Permission
	S7	Suspicious API calls
	S8	Network address

2.3 Limitations and open issues

Static analysis, however, has obvious limits because it cannot decipher malicious code that is downloaded, decrypted in-place, or highly obfuscated[11][12][13]. The idea is that the altered malware samples should not only be difficult to detect, but also difficult to trace back to the adversarial manipulation that caused them. Then, we give a thorough analysis of these Drebin attacks along with a brand-new learning technique to lessen their practical effects.

3 Attack models and scenarios

We use a previously defined attack model to conduct a complete security assessment of learning-based malware detection systems. which divides probable machine-learning algorithm threats into three categories based on security breach, attack specificity, and attack influence.

4 Types of attacks

The type of attacks depends on the attacker's goal, knowledge capability, strategy.

4.1 Evasion Attacks

An attack with the goal of avoiding detection or getting past security measures like firewalls, intrusion detection systems, antivirus software, or other security mechanisms is known as an evasion attack. An evasion attack aims to deceive the security system into granting the attacker unrestricted access to a network or system or the ability to execute harmful code covertly. Here the attacker's goal is violating the system integrity. It can be defined as:

$$z^{\star} = \arg\min_{z' \in \Omega(z)} \hat{f}(\Phi(z')) = \arg\min_{z' \in \Omega(z)} \hat{w}^{\top} x'$$

4.1.1 Zero-Effort Attacks

This is the typical situation where malware data is neither altered nor obscured in any way. This scenario is defined by an empty knowledge-parameter vector from the attacker's point of view.

4.2.2 DexGuard-Based Obfuscation Attacks

Here the attacker does not exploit any knowledge of the attacked system. Using the paid Android obfuscation tool DexGuard, the attacker attempts to avoid detection by conducting invasive code modifications on the classes.dex file. It should be noted that this program is not intended to conceal the presence of dangerous code, but rather to ensure protection against attempts to disassemble/decompile applications.

4.2.2 Mimicry Attack

A mimicry attack is a kind of cyberattack in which a hacker poses as a trusted user, device, or system in order to enter a network or system without authorization. In a mimicry attack, the attacker impersonates legitimate communications in order to evade detection by using stolen or forged credentials, such as usernames and passwords, or by taking advantage of security flaws in the target system. Moreover, it is important to note that this mimicking attack is more complex than those that are frequently employed in practice, where an attacker is typically thought to have masked a malicious program with a benign one [14][15].

4.2.3 Limited-Knowledge (LK) attacks

In this case the attackers understand the learning algorithm "l" which was used by the targeted system. You can use the available data to learn a surrogate classifier.

4.2.4 Perfect-Knowledge (PK) attacks

In the worst-case scenario, the attacker is also aware of the targeted classifier. Although though it is unlikely to occur, the attacker could learn the parameters of the trained classifier. This parameter is especially interesting since it gives the system under attack an upper bound on the performance degradation it will experience and serves as a benchmark for assessing the system's performance in other simulated attack scenarios.

4.3 Malware Data Manipulation

There are mainly two main settings in our evaluation which are below:
Feature Addition: The attacker can individually inject every feature in this situation.
Feature Addition and Removal: This hypothetical situation represents a more potent attacker who can both add and delete features from the dexcode.

The Android uses a verification system to monitor the integrity of an application while it is running (for example, it will terminate the application if a register passed as a parameter to an API call contains the incorrect type), and the likelihood that these actions will be compromised increases if features are carelessly deleted.

5 Adversarial Detection

In this section, we introduce an adversary -aware approach to improve the robustness of Drebin against the data manipulation attacks. So, for Drebin we develop a simple, light weight and scalable approach. This is why using computationally intensive learning processes along with non-linear classification functions is inappropriate for our application context. As a result, we have chosen to develop a linear classification method with enhanced security attributes, as explained in the sections that follow.

6 Methodology

6.1 Securing Linear Classification

From the previous work [16][17], our aim is to improve the security by adding more evenly-distributed feature weights as it increases the security and inherently require the attacker manipulating more features in order to avoid detection. Here we Our *classifier sensitivity* as

$$\Delta f(x, x') = \frac{f(x) - f(x')}{\|x - x'\|} = \frac{w^\top (x - x')}{\|x - x'\|} ,$$

which evaluates the decrease off when a malicious sample x is manipulated as x', where the maximum is reached when just one weight is not null, validating the notion that more evenly-distributed feature weights should increase classifier security under attack, and the minimum is reached for equal absolute weight values (independent of the number of alterations made to x). In comparison to heuristic techniques, our method is not only able to discover more equally distributed feature weights. Although it can outperform them in terms of security as well.
It's also important to note that our method maintains the convexity of the objective function that the learning algorithm optimized. This allows us to derive training methods with (perhaps strong) convergence guarantees that are computationally efficient.

6.2 Secure SVM learning algorithm

Here in this algorithm, we define the upper and lower bounds by the vectors w(lb) and w(up) which is selected with a suitable procedure. Here we use the SGD which is a light weight-based algorithm which is used for efficient learning on very large-scale datasets. Using a single sample or a small portion of the training data, randomly selected at each iteration, the objective function's sub gradients[18][19].The hinge loss and regularisation are merely two factors that our Sec-SVM learning algorithm considers[20][21]. This study results in our finding that there is an inherent trade-off between security and sparsity: while a sparse learning model ensures an efficient description of the learnt decision function, it may be quickly and easily defeated by merely changing a few features. A secure learning model, on the other hand, depends on the inclusion of numerous, possibly redundant features that make it more difficult to escape the decision function, but at the expense of a dense representation.

6.3 Parameter Selection

As indicated in, when adjusting the classifier's parameters, one should consider other factors besides only accuracy when using cross validation or bootstrapping to assess performance. Instead, one should account for the possibility of hidden attacks during the validation process to optimize the trade-off between security and accuracy. Here A is the measure of classification accuracy in the absence of attack with S is an estimate of the classifier security under the attack with lambda is a given trade -off parameter.

$$\boldsymbol{\mu}^\star = \arg\max_{\boldsymbol{\mu}} r(f_{\boldsymbol{\mu}}, \mathcal{D}) = A(f_{\boldsymbol{\mu}}, \mathcal{D}) + \lambda S(f_{\boldsymbol{\mu}}, \mathcal{D})$$

$$S = \frac{1}{\mathsf{M}} \sum_{m=1}^{\mathsf{M}} A(f_{\boldsymbol{\mu}}, \mathcal{D}'_k)$$

7 Observations and analysis

In this section, we present the results of a practices require of our proposed secure learning algorithm (Sec-SVM), which was put to the test in various evasion scenarios.

We contrast our Sec-SVM strategy with the common Drebin implementation (abbreviated SVM) and with a previously reported method that increases the security of linear classifiers by using a Multiple Classifier System architecture to produce a linear classifier with more uniformly distributed feature weights. The classifiers are then integrated by averaging their results, which is comparable to employing a linear classifier whose weights and bias are, respectively, the averages of the weights and biases of the basis classifiers. By using this straightforward technique, the computational complexity at test time is kept to the level of a single linear classifier. We refer to this strategy as MCS-SVM because linear SVMs serve as the basis for our classifiers. Finally, we explore a Sec-SVM variant of our Sec-SVM trained just on manifest features (M). The goal is to determine whether focusing primarily on attributes that cannot be changed prevents malicious data from closely resembling benign data, and whether this results in a more secure system [16].
Datasets. Here we use two data sets, the first one is drebin which includes the data in [8], which consists of 121,329 benign applications and 5,615 malicious samples labeled with Virus Total service and the other is contagio dataset yielding about 1500 samples.
Training-Test Splits. We split the random 60,000 samples into two equal sets of 30,000 samples each and use them as training

set and surrogate set and for test set, we use the remaining samples from Drebin. In some attack setups, we swap out the Drebin malware data with Contagio malware samples in each test set. This enables us to assess how well a classifier (trained on some data) maintains its effectiveness in identifying malware from several sources.
Feature Sets: As we are running Drebin on the given data sets more than one million of features are found.

TABLE 2
Number of Features in Each Set for SVM, Sec-SVM, and MCS-SVM

		Feature set sizes			
manifest	S_1	13 (21)	dexcode	S_5	147 (0)
	S_2	152 (243)		S_6	37 (0)
	S_3	2,542 (8,904)		S_7	3,029 (0)
	S_4	303 (832)		S_8	3,777 (0)

7.1 Experimental Evaluation

As the attacks discussed above(see section 4) we present the results by reporting the performance of the classifiers.
a) *Zero-Effort Attacks: Here* we consider two cases (i) using both test and training on debrin (ii) training and testing on contagio. Here MCS-SVM gets the highest DR which was followed by SVM, SEC-SVM which only slightly worse than DR Sec-SVM(M).
b) *DexGuard-Based Obfuscation Attacks*: The classifiers trained on drebin against the DexGuard-based Obfuscation attacks on the contagio malware. While Sec-SVM (M) and SVM normally have lower detection rates, Sec-SVM (M) here performs comparable to MCS-SVM. As expected, techniques like trivial, string encryption, and reflection have little to no impact on the system's speed because Drebin only considers system-based API calls, which are unaffected by those mentioned obfuscations. Class Encryption is the most effective defence against these attacks.

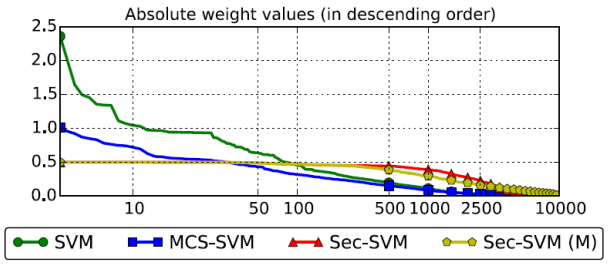

Fig.1 Absolute weight values for each classifier

Advanced Evasion: Comparing Sec-SVM to other classifiers, security can be significantly increased since Sec-SVM performs more gracefully against a growing number of modified features, particularly in the PK and LK attack scenarios. Finally, it is also important to note that mimicking attacks are less successful, as would be predicted given that they take use of a lower amount of system knowledge.
Feature Manipulation: However, the likelihood of changing a feature depends on both its likelihood of appearing in malware data as well as the weight assigned by the classifier, as was previously indicated. For example, if a (non-manifest) characteristic is present in every sample of malware and it has been given a very high positive weight, it will always be eliminated; on the other hand, if it only occasionally appears in malware, it will only be erased from a small number of samples. This behavior is clearly exhibited by the top features modified by Sec-SVM. The most often modified characteristics depend on the data distribution, not just the likelihood that they would appear more frequently in malware (e.g., class imbalance, feature correlations, etc.), in a clear way.
Robustness and Regularization: Interestingly, a recent theoretical justification for the need to alter more features in order to defeat our Sec-SVM can also be found in [22].
Robust optimization problem with worst-case (spherical) noise-corrupted input data. Due to its tendency to somewhat affect all feature values, this noise should be regarded dense.
How ever there are some restrictions and issues like, It is shown when our evasion attempts change a lot of aspects, especially when mimicry attacks are used and the changed malware samples almost identically imitate benign data. (in terms of their feature vectors). Again, this is conceivable due to a vulnerability in the feature representation that is inherent, and no learning system can accurately and clearly distinguish such data. Another drawback of our method may be its unsatisfactory performance under PK and LK attacks, but this is easily remedied with straightforward countermeasures to stop the attacker from learning enough about the attacked system, like routine system re-training and diversification of training data collection.

8 CONCLUSIONS AND FUTURE WORK

The target of our work is to demonstrate how machine learning may be utilized to enhance system security, provided one adopts an adversary-aware strategy that proactively expects the attacker. This study key contribution is the definition of a novel, theoretically sound learning algorithm for the training of linear classifiers with more equally distributed feature weights. Without significantly compromising computational efficiency, this method enables one to increase system security (in terms of the number of careful alterations to the malware samples required). Extending our method for secure learning to nonlinear classifiers, such as using nonlinear kernel functions, is a possible extension of our work that may further enhance classifier security. Recent study on the resilience and regularization of learning algorithms may offer creative new possibilities.
Instead, the learning algorithm may be modified to incorporate the knowledge of categorizing features according to their robustness to adversarial manipulations, with larger (absolute) weight values being given to more robust characteristics.

Finally, it is important to note that we have recently used the suggested learning technique to strengthen the protection of JavaScript and PDF malware detection systems against sparse evasion assaults [23][24]. This demonstrates that our proposal not only gives a first, practical example of how machine learning can be used to increase the security of Android malware detectors, but also that our design methodology can be easily adapted to additional learning-based malware detection jobs.

REFERENCES:

[1] I. Arce, "The weakest link revisited [information Sec.]," IEEE Secur. Privacy, vol. 1, no. 2, pp. 72–76, Mar. 2003.
[2] B. Biggio, et al., "Evasion attacks against machine learning at test time," in Proc. Eur. Conf. Mach. Learn. Principles Practice Knowl.Discovery Databases, 2013, pp. 387–402.
[3] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in Proc. 29th Int. Conf. Mach. Learn., 2012, pp. 1807–1814.
[4] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in Proc. 9th Int. Conf. Recent Advances Intrusion Detection, 2006, pp. 81–105.
[5] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading worm signature generators using deliberate noise injection," in Proc. IEEE Symp. Privacy, 2006, pp. 17–31.
[6] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao, "Man versus machine: Practical adversarial detection of malicious crowdsourcing workers," in Proc. 23rd USENIX Secur. Symp., 2014, pp. 239–254.
[7] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Isfeature selection secure against training data poisoning?" in Proc. 32nd Int. Conf. Int. Conf. Mach. Learn., 2015, vol. 37, pp. 1689–1698.
[8] D. Arp, M. Spreitzenbarth, M. H€ubner, H. Gascon, and K. Rieck,"Drebin: Efficient and explainable detection of android malware in your pocket," in Proc. 21st NDSS, 2014, p. 12.
[9] M. Br€uckner, C. Kanzow, and T. Scheffer, "Static prediction gamesfor adversarial learning problems," J. Mach. Learn. Res., vol. 13, pp. 2617–2654, Sep. 2012.
[10] A. Kolcz and C. H. Teo, "Feature weighting for improved classifier robustness," in Proc. 6th Conf. Email Anti-Spam, 2009, p. 8.
[11] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Dept. Comput. Sci., Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, Jul. 1997.
[12] V. Rastogi, Z. Qu, J. McClurg, Y. Cao, and Y. Chen, Uranine: Real-Time Privacy Leakage Monitoring Without System Modification for Android. Cham, Switzerland: Springer, 2015, pp. 256–276.
[13] W. Yang, et al., AppSpear: Bytecode Decrypting and DEX Reassembling for Packed Android Malware. Cham, Switzerland: Springer,2015, pp. 359–381.
[14] N. _Srndic and P. Laskov, "Practical evasion of a learning-based
classifier: A case study," in Proc. IEEE Symp. Privacy, 2014, pp. 197–211.
[15] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in Proc. IEEE Symp. Privacy, 2012, pp. 95–109.
[16] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments," Int. J. Mach.Learn. Cybern., vol. 1, no. 1, pp. 27–41, 2010.
[17] A. Demontis, P. Russu, B. Biggio, G. Fumera, and F. Roli, "On Sec. and sparsity of linear classifiers for adversarial settings," in Proc.Joint IAPR Int. Workshop Structural Syntactic Statist. Pattern Recognit., 2016, pp. 322–332.
[18] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in Proc. 21st Int. Conf. Comput. Statist., 2010, pp. 177–186.
[19] T. Zhang, "Solving large scale linear prediction problems using SGD algorithms," in Proc. 21st Int. Conf. Mach. Learn., 2004, pp. 116–123.
[20] C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol. 20, pp. 273–297, 1995.
[21] V. N. Vapnik, The Nature of Statistical Learning Theory. Berlin, Germany: Springer, 1995.
[22] H. Xu, C. Caramanis, and S. Mannor, "Robustness and regularization of support vector Machines," J. Mach. Learn. Res., vol. 10, pp. 1485–1510, Jul. 2009.
[23] A. Demontis, P. Russu, B. Biggio, G. Fumera, and F. Roli, "On Sec. and sparsity of linear classifiers for adversarial settings," in Proc. Joint IAPR Int. Workshop Structural Syntactic Statist. Pattern Recognit., 2016, pp. 322–332.
[24] P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli, "Secure kernel machines against evasion attacks," in Proc. 9th ACM Workshop Artif. Intell. Secur., 2016, pp. 59–69.
[25] Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection Ambra Demontis, Student Member, IEEE, Marco Melis, Student Member, IEEE, Battista Biggio, Senior Member, IEEE, Davide Maiorca, Member, IEEE, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, Senior Member,IEEE, and FabRoli,Fellow,IEEE 02 May 2017 Pages 711-724.