Adversarial Filter on Malware Domain

Rafael Gomes Moreira[1], Rafael Oliveira da Rocha[1], Idilio Drago[2]e Lourenço A. Pereira Jr.[1]
[1]Instituto Tecnológico da Aeronáutica, SJC/SP - Brasil
[2]University of Turin, Turin - Itália

***Abstract*— This research investigates the feasibility of employing adversarial filtering techniques to counter evasion attacks on a Support Vector Machine malware detection model. Exploring the feasibility of packed use and the potential differences in performance when incorporating the adversarial classifier before or after the primary model. The research aims to demonstrate the efficient operation of the models on a host machine while utilizing medium to lower computational resources.**

***Keywords*— Malware, Cybersecurity, Machine Learning.**

I. Introduction

Machine Learning has become an excellent tool for classifying malware areas, bringing about a significant transformation in this domain, principally because of its generalization power when working with a lot of data. However, with new attacks aimed at circumventing these protective measures, the focus has shifted to the targeted system and the methodologies employed during the classification to safeguard against such threats. Adversarial attacks specifically exploit the vulnerabilities in these methodologies[1][2], making them fragile to misclassification. These attacks involve deliberately introducing adversarial samples or some kind of noise[3] into the training dataset or manipulation of malware features to evade the protection mechanisms[4].

Moreover, given the paramount significance of information security and the pervasive presence of mobile devices, particularly within households where at least one cell phone is typically found, it becomes imperative to address potential vulnerabilities. In the present era, cell phones are a multifunctional tool enabling individuals to manage their financial transactions, store personal data, and access sensitive information. Consequently, any breach in security concerning these devices necessitates a thorough investigation.

Adversarial attacks, posing a significant threat, have emerged as a novel approach to circumvent existing malware protection mechanisms[5]. These attacks capitalize on the utilization of known malicious artefacts and exploit specific scenarios to achieve their objectives without being detected. Considering the evolving landscape of security threats and the heightened sophistication of adversaries, it is crucial to delve into the study and understanding of adversarial attacks in the realm of malware detection.

Throughout our research, we investigated existing frameworks that incorporate an adversarial classifier as a system component. Notably, prior works have utilized an adversarial classifier either within an ensemble model or as a component[6]. However, none have employed the adversarial classifier as a direct filter to assess the efficacy of this novel system.

R.G. Moreira, moreirargm@ita.br; R. O. da Rocha, rafaelror@ita.br; Idílio Drago, idilio.drago@unito.it; L. A. Pereira Jr, lourenco.junior@gp.ita.br.

Our study focused on integrating a classifier model directly into a malware classifier for the Android Operating System. We specifically examined performance variations from incorporating the adversarial classifier before or after the primary model. We aimed to analyze these variations and gain insights into how integrating the adversarial classifier at different stages of the model architecture impacts its effectiveness.

A. Our Contributions

In this paper, we make the following contributions. First, we evaluate the precision and recall of the SVM model used for malware classification and the variation of that metric values when adversarial samples are injected. Secondly, we assess the time required to train both the primary classification model and the adversarial filter model. Additionally, we explore the feasibility of utilizing an ensemble or combining the primary classification model and the adversarial model to operate efficiently on a host machine, such as a conventional personal computer, while optimizing the utilization of medium to lower computational resources.

II. Background Knowledge

A. Adversarial Examples

Some machine learning algorithms' inputs are designed to confound their functionality and increase the likelihood of misclassification[7]. For instance, a machine learning algorithm classifies an image as a "table class". We can generate some noise and inject it into that image to change the classification of the same algorithm to a "door class".

B. Problem Space and Feature Space

When dealing with artifacts such as malware or images, we need to transform them into feature vectors. The Problem Space comprises problems where we manipulate feature vectors without considering the transformation of real-world artifacts. The Problem Space becomes evident when mutations have functional implications in the real world. This means an artifact is transformed into a feature vector, altered with noise, and then transformed back into a fully functional real-world artifact[8].

C. Inverse Feature-Mapping Problem

Transitioning from Feature Space to Problem Space becomes complex when features change. As most attacks involve changes in the Feature Space, converting these new features back into fully functional code poses a significant challenge when dealing with adversarial attacks[9].

D. Evasion Attacks

Evasion Attacks occur when changes are made to the artifact's structure to replace the original features' information with other data, introducing perturbations or injections. This alteration aims to confuse the machine learning model's classification. These attacks can take place in either Feature or Problem Space. However, it's crucial to note that they are subject to constraints specific to the domain of the artifact. For example, if an evasion attack is performed on a malware artifact, it should demonstrate that specific manipulations can compromise the application's integrity or malicious functionality[10].

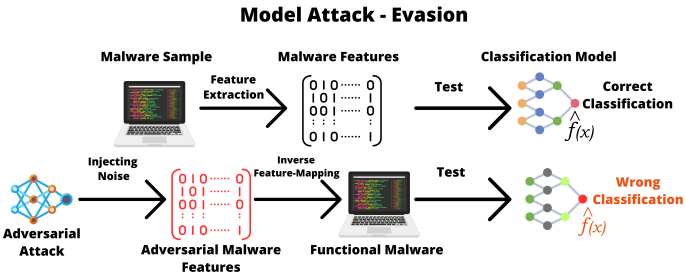

Fig. 1. Evasion Attack.

As we can see in Fig. 1, in a generalist classifier model using machine learning to detect malware, we must convert a real-world artefact to a feature-space artefact represented by a vector of features. This vector of features is an input to a classification model, and this one answers their class. When we have an Evasion Attack, the technique (Mimicry, AT-MA or others) creates some perturbation in that feature till the classification model points to the wrong class.

E. Poisoning Attacks

Machine learning models rely on the assumption that training data exhibits particular statistical properties, enabling accurate predictions. However, this assumption opens the door to false presumptions and experimental biases[11], especially when training data is manipulated through techniques like poisoning attacks.

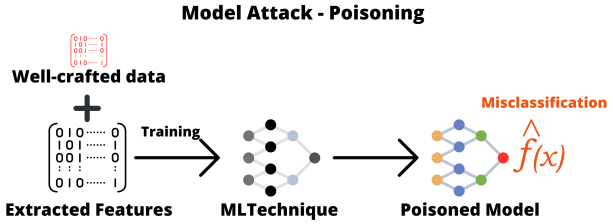

Fig. 2. Poisoning Attack.

Poisoning attacks introduce malicious or adversarial samples into the training dataset to deceive the model during learning. By strategically altering the training data, adversaries can exploit the model's reliance on statistical properties and lead it to make incorrect or biased predictions[8].

As we can see in the Fig. 2, if we inject some well-crafted data in the extracted features used to train the malware classifier model, we could create some specific errors during the training phase of the model, and it could be used to baffle the classification during the test phase of a malware. That way, the adversarial attack can benefit a lot of malware class, despite the evasion attack, which benefits only that specific malware.

III. Related Works

We review adversarial attack methods from the defender's perspective, focusing on preview works that used a system with or without adversarial classifier versions. [6] uses an adversarial classifier as part of an Ensemble Model, but defining the rule of results combination as an ensemble model is essential. In [10], the research uses the adversary detector as an auxiliary ML model and uses it after the malware detector. Furthermore, as an adversary detector, they use a Deep Neural Networking based Machine Learning Classifier Model, making processing more costly on a personal computer. In [9], the authors proposed a novel of attack and a way to realize the specific inverse feature-mapping, using the weight of the functions and their correspondence with the features. In [12], they create feature-extraction rules of android APK applications and elaborate a classifier system using an SVM model. The features extracted from Drebin are static, and there are many ways to bypass static feature classifiers during the execution time. However, the static features are a good resource because the most preview works in our scope, the adversarial attacks changed the characteristics of the artefact, which is detected in static feature-extraction.

IV. Experiments and Evaluation

A. Experimental Setup and premises

To conduct our experiment, we established specific premises to ensure the validity of our approach. In the context of protection, we assumed that the Inverse feature mapping was resolved in the adversarial malware being used. This implies that the samples we considered are fully functional in terms of their malicious behaviours. Additionally, we treated our data as Independent and Identically Distributed (IID), ensuring that there was no data leakage during both the training and testing phases of our models.

To simulate a Personal computer, it was used a computer with i7 7th generation with 16 GB of ram.

The malware and goodware samples were downloaded from Androzoo API[13], and the feature-extraction was made using the definitions of Drebin Dataset[12] and the implementations presented in [6].

1) Feature Extraction: In this work, we used a Feature-extraction based on Drebin Dataset[12]. Their feature representations are based on information about permissions, API calls, intent filters, string analysis and code analysis of an Android's APK file.

Permissions: The analysis of the requested permissions by the application, declared in the AndroidManifest.xml.

API calls: Analysis of the interaction with the Android system and external resources (API calls).

Intent Filters: Investigation of the types of intents the application can handle and that used to exploit the system.

String Analysis: Examination of strings inside the application's code and their resources. Specific keywords, obfuscation or encryption may be identified through string analysis.

Code Analysis: The analysis of code's structure, control and data flow.

All of that information was converted into a matrix of features. The most important information is defined when their matrix is changed into a TF-IDF matrix, keeping only the recurrent information during the analysis[14].

2) Scenarios and Data: The evaluate the premises, it was generated adversarial attacks against a basic DNN classifier model using a Projected Gradient Descent(PGD) with Adam Optimization[15](AT-MA) and a Mimicry attack[16]. We used 800 malware samples downloaded from Androzoo API to generate the adversarial samples.

Gradient-Based Attack: It is a white-box attack (it needs to know the target classifier) that uses the gradient of the classifier's loss function.

Mimicry Attack: It is a gradient-free attack and occurs when we perturb a malware example until it mimics the goodware (benign sample) behaviour.

A Support Vector Machine (SVM) Technique is trained to create the classifier model with 16.000 malware samples and 20.000 goodware samples. The experiments were made with SVM model because of their ability to handle high-dimensional and not linearly separated data. Furthermore, despite the dataset being balanced, malware datasets normally are not balanced in the wild, and we used a model that performs well with unbalanced data.

B. Evaluating the results acquired

1) Basic training of the Model: During this phase, we have trained the Classifier model (SVM), with 16.000 malware samples and 20.000 goodware samples, using the squared hinge loss function. To that work, using the setup referenced, it costs 7.54 seconds of training and the following results:

TABLE I

RESULTS - TRAINING MAIN MODEL

	Precision	Recall	F1-score
Malware	0.83	0.93	0.88
Goodware	0.92	0.80	0.86
Avg / Total	0.8691	0.87	0.87

After that, we trained the Adversarial Classifier Model we wanted to test with the primary classifier. To build that model, we used an SVM technique trained with 800 adversarial malware focused on Deep Neural Network classifier models and 780 non-adversarial malware to create the adversarial ones, and we used a split of 30%. That dataset was modelled to harden the detection of adversarial samples despite the non-adversarial ones. The time used for its training was 1.12 seconds.

So, we have made the same before but focused on AT-MA adversarial attacks. That dataset was modelling trying to harden the detection of adversarial samples despite the non-adversarial ones. The time used for its training was 1.22 seconds.

TABLE II

RESULTS - TRAINING FILTER MODEL TO MIMICRY ATTACKS ADVERSARIAL SAMPLES

	Precision	Recall	F1-score
Malware Non Adversarial	1.0	0.98	0.99
Malware Adversarial (Mimicry attack)	0.98	1.0	0.99
Avg / Total	0.9916	0.99	0.99

TABLE III

RESULTS - TRAINING FILTER MODEL AGAINST AT-MA ADVERSARIAL SAMPLES

	Precision	Recall	F1-score
Malware Non Adversarial	1.0	0.99	1.0
Malware Adversarial (AT-MA target)	0.99	1.0	1.00
Avg / Total	0.9957	1.0	1.0

After that phase, when we trained the models and compared the result for the primary classifier and the adversarial classifiers, we started the experiment's second phase, testing the models' detection against adversarial samples.

TABLE IV

DETECTION OF ADVERSARIAL SAMPLES (DNN TARGET)

	Precision	Recall	F1-score
Malware Adversarial (DNN target)	0.77	0.62	0.69
Goodware	0.63	0.78	0.69
Avg / Total	0.71	0.69	0.69

TABLE V

DETECTION OF ADVERSARIAL SAMPLES (AT-MA TARGET)

	Precision	Recall	F1-score
Malware Adversarial (AT-MA target)	0.81	0.79	0.8
Goodware	0.75	0.78	0.77
Avg / Total	0.9957	01.0	1.0

The last phase of experiments tested the adversarial samples (Mimicry and AT-MA) and Non-Adversarial Malware in the main classifier. The objective of that experiment phase was to verify the behaviour of the main classifier when it needed to classifier malware, and there were adversarial samples in there. During that phase, we had the following results:

TABLE VI

TEST OF ADVERSARIAL MALWARE(MIMICRY) AND NON-ADVERSARIAL MALWARE AGAINST THE MAIN CLASSIFIER MODEL

	Precision	Recall	F1-score
Non Adversarial Malware	0.54	0.74	0.62
Adversarial Malware(Mimicry)	0.40	0.62	0.53
Avg / Total	0.47	0.68	0.58

TABLE VII

TEST OF ADVERSARIAL MALWARE(AT-MA) AND NON-ADVERSARIAL MALWARE AGAINST THE MAIN CLASSIFIER MODEL

	Precision	Recall	F1-score
Non Adversarial Malware	0.48	0.74	0.58
Adversarial Malware (AT-MA)	0.55	0.79	0.71
Avg / Total	0.52	0.77	0.65

V. Discussion

With the preliminary results, we observed that an SVM model trained solely on Adversarial and non-Adversarial malware, used to generate the former, achieved nearly 100% precision in detection, as shown in Tables II and III. This observation is crucial because it confirms the premise of using an adversarial filter to cleanse the dataset of adversarial samples within the defined scope. Versions of the Mimicry attack, as well as AT-MA versions, were detectable with their respective adversarial filters. We observed that the malware classifier's precision dropped by 6% when classifying Adversarial samples employing the Mimicry technique, and by 2% when classifying samples using the AT-MA technique. While these figures might not appear low, in the realm of cybersecurity, a single unblocked malware instance could potentially compromise an entire system. Upon examining the results presented in table VI and VII, we can infer that utilizing the filter before or after the initial malware classification model yields equivalent final results. This approach proves superior to directly testing the primary model with the dataset containing adversarial and non-adversarial malware.

The training time as a primary model (malware classifier) as the filter models was below 12 seconds on average, with the defined scope. If the dataset enlarges in size, maybe another packing technique would be more viable. Still, in the defined scope, it is viable to run that model on a personal computer when we compare the time used to train the models.

When we check the results described in table VII, we can infer that using the filter before or after the malware classification model initially has the same final results, and it is better than only trying to test directly to the primary model the dataset with adversarial malware and non-adversarial malware.

VI. Conclusion and Future Works

In the realm of adversarial machine learning and malware detection, employing an adversarial filter in conjunction with a malware classifier, in the scope of this article, has proven to yield improved results. By incorporating an adversarial filter into the detection framework, the precision of the malware classifier is enhanced.

The addition of an adversarial filter introduces an extra layer of defense against adversarial attacks designed to manipulate or deceive the classifier. This filter identifies and mitigates the impact of adversarial samples within the training and testing datasets, effectively reducing their influence on the classification process. Furthermore, the adversarial filter assists in mitigating the risks associated with false positives or negatives, which can have grave consequences in the context of malware detection. By effectively identifying and addressing adversarial samples, the filter minimizes the likelihood of misclassifying benign applications as malicious or vice versa. Furthermore, exploring new testing models and comparing their processing times, as well as the required disk space for implementation, could yield significant insights. This becomes especially relevant in embedded computing scenarios where resource constraints are prevalent.

Integrating the adversarial filter with the malware classifier leads to several advantages. Firstly, it aids in detecting and neutralizing the effects of adversarial perturbations that could potentially mislead the classifier's decision-making, as we can see in tables VI and VII. The classifier becomes more resilient to adversarial manipulation by selectively filtering out such perturbations, as seen in I. Preview works used Ensemble Models, which necessarily uses more process and will probably need more time to train and test the models. In this work, we saw that the direct connection between the filter model and the malware classifier model already brings some gain in the classification precision.

Moreover, the adversarial filter helps mitigate the risks associated with false positives or false negatives, which can have severe consequences in the context of malware detection. By effectively identifying and addressing adversarial samples, the filter reduces the likelihood of misclassifying benign applications as malicious or vice versa.

These findings contribute to the ongoing efforts in adversarial machine learning and malware detection, providing valuable insights for developing more robust and effective defences against adversarial attacks. By integrating the protective capabilities of an adversarial filter directly into the primary classifier, their resilience could be strengthened, ensuring enhanced security in the face of sophisticated adversarial threats.

Future works can focus on enlarging the attacks that compose the adversarial dataset, principally when we mix attacks against different platforms, binary structures, etc.

Furthermore, the use of new testing models and the comparison between the processing, the time and disk space needed to implement them could be substantial, principally in embedded computing where there are low resources.

References

[1] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 387–402.

[2] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, 2014.

[3] N. Šrndić and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 197–211.

[4] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," 2018.

[5] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, ser. AISec '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 43–58. [Online]. Available: https://doi.org/10.1145/2046684.2046692

[6] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *Trans. Info. For. Sec.*, vol. 15, p. 3886–3900, jan 2020. [Online]. Available: https://doi.org/10.1109/TIFS.2020.3003571

[7] H. Rathore and S. K. Sahay, "Towards robust android malware detection models using adversarial learning," in *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, 2021, pp. 424–425.

[8] D. Li, Q. Li, Y. F. Ye, and S. Xu, "Arms race in adversarial malware detection: A survey," *ACM Comput. Surv.*, vol. 55, no. 1, nov 2021. [Online]. Available: https://doi.org/10.1145/3484491

[9] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1332–1349.

[10] D. Li, S. Cui, Y. Li, J. Xu, F. Xiao, and S. Xu, "Pad: Towards principled adversarial malware detection against evasion attacks," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–16, 2023.

[11] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/arp

[12] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," 02 2014.

[13] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 468–471. [Online]. Available: http://doi.acm.org/10.1145/2901739.2903508

[14] Q. Liu, J. Wang, D. Zhang, Y. Yang, and N. Wang, "Text features extraction based on tf-idf associating semantic," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 2018, pp. 2338–2343.

[15] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge," 2020.

[16] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, 2019.