Demystifying the Role of Rule-based Detection in AI Systems for Windows Malware Detection

Andrea Ponte*, Luca Demetrio*, Luca Oneto*, Ivan Tesfai Ogbu‡, Battista Biggio†, Fabio Roli*†

*University of Genova, Genova, Italy
andrea.ponte@edu.unige.it,
{luca.demetrio, luca.oneto, fabio.roli}@unige.it
†University of Cagliari, Cagliari, Italy
battista.biggio@unica.it
‡RINA Consulting S.p.A., Genova, Italy
ivan.tesfai@rina.org

***Abstract*—Malware detection increasingly relies on AI systems that integrate signature-based detection with machine learning. However, these components are typically developed and combined in isolation, missing opportunities to reduce data complexity and strengthen defenses against adversarial EXEmples, carefully crafted programs designed to evade detection. Hence, in this work we investigate the influence that signature-based detection exerts on model training, when they are included inside the training pipeline. Specifically, we compare models trained on a comprehensive dataset with an AI system whose machine learning component is trained solely on samples not already flagged by signatures. Our results demonstrate improved robustness to both adversarial EXEmples and temporal data drift, although this comes at the cost of a fixed lower bound on false positives, driven by suboptimal rule selection. We conclude by discussing these limitations and outlining how future research could extend AI-based malware detection to include dynamic analysis, thereby further enhancing system resilience.**

***Index Terms*—AI Systems, Malware Detection, Detection Pipeline, Adversarial Robustness.**

1. Introduction

To increase the likelihood of detecting the always-evolving variants of malware, malware detectors are enriched with several layers of detection, both relying on pattern matching and machine learning (ML) components. We name such combination as *AI systems*, aligned with the latest directives of the European Union through the EU AI Act [1]. This is consistently reported as the standard by companies that sell antivirus (AV) programs [2, 3, 4, 5], while academia is recently starting to investigate the problem of composing sequential layers of detection modules [6]. Regardless of their provenience, either from industry or academia, the first layer of detection is achieved through pattern-matching with YARA rules.[1] Manually crafted by domain experts, YARA rules contain both specific patterns of byte associated with malicious samples, and conditions that must be met to trigger detection.

1. https://github.com/VirusTotal/yara

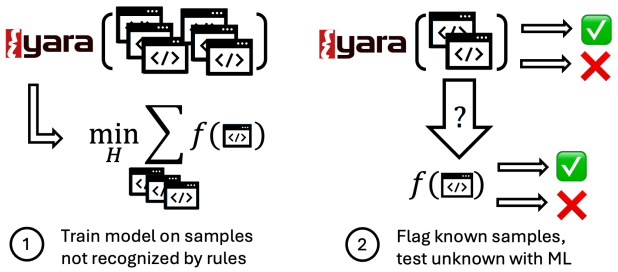


Figure 1: AI System for malware detection, which combines signature-based detection to ML, both during (1) training, by removing samples that trigger YARA rules, and (2) test time, by using ML only on unknown samples.

However, even if many samples are already stopped by available rules, malware detectors are trained on all available data, and deployed alongside YARA rules. While intuitive, this methodology might not be optimal, since each layer of detection alters the distribution of data fed in input to the subsequent detection module. As a result, models are trained on samples they will never see at test time, complicating not only the training process due to the massive amount of samples used to create these detectors, but also potentially including spurious correlations exploited by attackers with *adversarial EXEmples* – carefully-manipulated Windows malware that try to evade ML detection [7, 8].

Hence, in this work we perform preliminary steps towards an empirical understanding of the role of YARA detection layer at training and test time, as shown in Fig. 1. As far as we know, we are the first to propose such an analysis, where we exclude from training data *all* those samples that are either matched by at least one YARA rule, or are contained inside an allowlist composed utilities harvested from fresh installation of Windows. We empirically quantify how such pre-filtering influences the learning process of the AI system we build, showing that this methodology matches and exceeds the performances of models trained on all the available samples at low FPR (1%), also when exposed to never-seen future data possi-

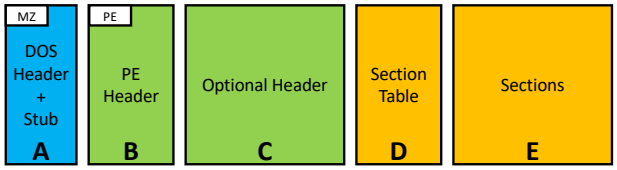


Figure 2: Depiction of the Windows PE File Format.

bly affected by concept drift. Also, thanks to the presence of YARA rules, the AI system we build exhibits higher robustness to adversarial EXEmples rather then regularly-trained models, since the decision function is harder to explore by the attack. Similar to previous work [6], we observe that the computation of adversarial EXEmples can introduce unforeseeable artifacts that are detected by rules when injected by the optimization process. However, these benefits are balanced by a fixed amount of false positives induced by rules themselves. Even if some of them are very effective in stopping hundreds of malware samples, they can not be tuned to adapt to the training data and reduce their response to false alarms, differently from machine learning models. We conclude our work by remarking that these are preliminary results on the performance of AI systems for malware detection, but already highlighting potential improvement in the field in terms of efficiency and robustness.

2. Background and Related Work

Before describing our methodology, we introduce the key concepts needed to fully understand our manuscript, comprising the types of data we are dealing with, how to detect malicious samples among them, and how attackers can evolve their technique to evade complex ML models used for malware detection.

Windows PE File Format. Each Windows program is stored as a file, whose structure is defined by the Portable Executable (PE) format[2]. As depicted Fig. 2, the format provides precise information on how to load programs in memory, and it is divided into metadata and code. The latter, alongside other relevant information such as initialized data, are stored in multiple *sections* (E in Fig. 2), which constitute the majority of each program.

Detection with YARA Rules. YARA is a pattern-matching tool used to detect known malware based on signatures. These signatures are textual descriptions containing binary patterns that help identify malicious programs. Each YARA rule consists of metadata that describes its purpose, followed by strings or patterns—such as hexadecimal sequences or regular expressions—that the tool searches for within files. Finally, each rule includes a firing condition, which determines how malicious activity is identified. This condition is structured as an if-then-else logic block that defines the detection algorithm. This mechanism is not only helpful in detecting malware samples, since conditions can be designed to identify well-known unharmful executables, such as operating systems applications and certified software installers.

Static Malware Detection with Machine Learning. This type of analysis discriminates legitimate and malicious programs by analyzing the sole structure of executables without running them, and train machine learning models on top of these activities. The latter is achieved by training models either on features, which are measurements computed on data through dedicated algorithms that leverage domain knowledge [9, 10]; or on raw bytes [11] often treated as gray-scale images [12]; or on a combination of the two mentioned approaches [13]. While promising, static analysis might be circumvented by obfuscation since malware samples can manifest malicious behaviors at runtime that cannot be easily inferred without execution. Also, complex feature extraction algorithms might crash on specific samples, requiring flexibility in the analysis process. To overcome these limitations, static malware detectors can be improved by either applying redundant and multiple controls [6] also involving signature-based detection, or merging it with *dynamic* analysis [14], which also captures the runtime behavior of samples. However, as far as we know, none of these work on either single- or multi-layer detectors considered the influence that each layer imposes on the next. In fact, no research work has investigated the effect of the change of data distribution between layers, and how this affects the final performances in production.

Adversarial EXEmples. As previously shown in literature, Windows malware detectors can be ineffective against *Adversarial EXEmples*, carefully-manipulated programs that evade detection without altering their original functionality [7, 8, 15]. These samples are crafted through optimization algorithms that only rely on the answer of the target model to evade, manipulating the samples through content-injection procedures. To improve the likelihood of success, there are techniques like GAMMA [15] that fools the target model by injecting into malware sample content extracted from legitimate programs, without altering the flow of execution.

3. Experimental Analysis

Our methodology revolves around removing samples from the training data that are already detected by predefined YARA rules, and then we train a machine learning model only on the undetected ones. Hence, we start by describing the setup (Sec. 3.1), and how the select dataset is filtered by YARA rules (Sec. 3.2). We empirically analyze the effect of such a filtering on the performances at test time in terms of accuracy and false positives (Sec. 3.3), and its robustness against attacks (Sec. 3.4).

3.1. Experimental Setup

In this subsection, we describe how we setup the experiments and which data we leverage to train and test the machine learning models.

Dataset. We use the Speakeasy dataset [14] as the main source of malware and goodware samples. It comprises several malware families as reported in [14] and is divided into training and test data. This dataset is unbalanced, and it counts 71506 malware samples and 26059 goodware samples.[3] To deal with such an imbalance, we increase the

2. https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

3. These numbers are slightly different from the ones reported by authors of the dataset, as we removed some duplicates we found before conducting our experiments.

number of benign samples in the training set by complementing it (i) with 9864 goodware samples collected from Chocolatey;[4] and (ii) with system files stored in *sys32* and *syswow64* of fresh installation of Windows 8.1, 10 and 11, for a total of 2648 goodware. Inside Chocolatey goodware samples, we found 54 Windows system files already present in our fresh installations, that we discard to avoid repetitions. We also add 5000 malware samples and 3446 goodware samples used in [15] and [6]. The Speakeasy dataset is characterized by a temporal shift between the training (collected in January 2022) and the test data (collected in April 2022), which is is useful for testing the performance of the models in an evolving scenario. Hence, we name our corpus of training data of Speakeasy and all the additions we have mentioned as Present Data (counting 76506 malware and 41952 goodware samples), and the Speakeasy test data (17500 malware and 10000 goodware) as Future Data. However, the goodware samples included in the training belong to the same time period of the ones in the test, due to the inclusion of the goodware samples from Chocolaty to improve the balance between classes. We are aware that such a setting could lead to *data snooping* [16], but such injection of goodware from other sources addresses the problem of the class imbalance. We measure the performances of models on Present Data building a test set composed of 10% of the total amount of samples (7685 malware and 4162 goodware). For brevity we indicate the Present test set as $\mathbf{T_p}$ and the Future test set as $\mathbf{T_f}$. For each file of the dataset, we compute the EMBER features [9], which represents a golden standard in static analysis feature extraction. The feature extraction failed for 4 samples, which we excluded from the datasets.

YARA Rules. We collect YARA rules for our blocklist from several public GitHub repositories[5,6,7,8,9], collecting 2.7K rules. As regards the allowlist, we created a rule containing the hashes of the Windows system files inside our dataset, that checks if the input is a well-known legitimate system program.

Selected Architectures. We select two ML models for our experiments: a Support Vector Machine [17, 18] (SVM) with Gaussian kernel implemented in SciKit-Learn [19] and a Gradient Boosted Decision Tree (GBDT) [20] implemented with the XGBoost library [21]. The best hyper-parameters γ and C for the SVM are chosen after an extensive model selection [22] choosing $\gamma \in 10^{\{-6.0,-5.5,...,4.0\}}$ and $C \in 10^{\{-6.0,-5.5,...,4.0\}}$, trying 30 values spaced evenly on a log scale, for both parameters. We perform a grid-search cross-validation with 10 folds, and we use a subset of our large training with 30K samples for each class. The resulting values for the hyper-parameters are used for both the regularly-trained models and the AI system trained on filtered data. For GBDT we set 1000 trees and $\eta = 0.1$, the subsample ratio of columns when constructing each tree to 0.8, while keeping default values for all the other parameters. We produce models trained only on data that do not trigger any YARA rule

4. https://community.chocolatey.org/
5. https://github.com/bartblaze/Yara-rules/tree/master/rules
6. https://github.com/elastic/protections-artifacts/tree/main/yara/rules
7. https://github.com/malpedia/signator-rules/tree/main/rules
8. https://github.com/Neo23x0/signature-base/tree/master/yara
9. https://github.com/Yara-Rules/rules/tree/master/malware

$\mathbf{D_t}$		$\mathbf{T_p}$		$\mathbf{T_f}$	
TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
0.24	0.0094	0.23	0.0096	0.34	0.0089

TABLE 1: YARA performance (represented by TPR and FPR) on the training set $\mathbf{D_t}$, the present test set $\mathbf{T_p}$ and the future test set $\mathbf{T_f}$.

denoted as SVM_f and GBDT_f, and we compare them with models trained on all the data we possess (noted as SVM and GBDT).

Setup of Adversarial Attacks. The robustness evaluation of all the models we train is performed with GAMMA, that leverages the injection of new sections harvested from legitimate programs [15] to manipulate malware. The attacks are conducted on a subset of the test set $\mathbf{T_p}$ and on a subset of $\mathbf{T_f}$, randomly selecting 100 malware samples from each family in the Speakeasy dataset (resulting in a total of 700 samples gathered from both sets). Since we are interested in the effect imposed by rules filtering, which considers different data distributions at training time, we setup GAMMA to harvest content from two sources: (i) legitimate Windows utility programs extracted from a fresh installation of Windows 10; and (ii) samples scraped from Chocolatey[10] which is a well-known package manager for Windows. In both cases, we test attacks that can query the target at maximum 200 times, with increasing forces, allowing GAMMA to select content harvested from 10, 20, 30, and 50 sections among these legitimate programs. This procedure leads to 32 attack configurations for both models. For the SVM models, we fix the regularization term $\lambda = 10^{-5}$ and the number of queries to 200 for all configurations. For the GBDT models we fix the number of queries to 500, and we set $\lambda = 10^{-7}$ when using 10 to 20 sections extracted from Windows goodware, while $\lambda = 10^{-8}$ for 30 and 50 sections from the same source. Instead, when using Chocolatey PEs which are bigger in size, we set $\lambda = 10^{-6}$ for 10 and 20 sections attacks and $\lambda = 10^{-7}$ for 30 and 50 sections attacks.

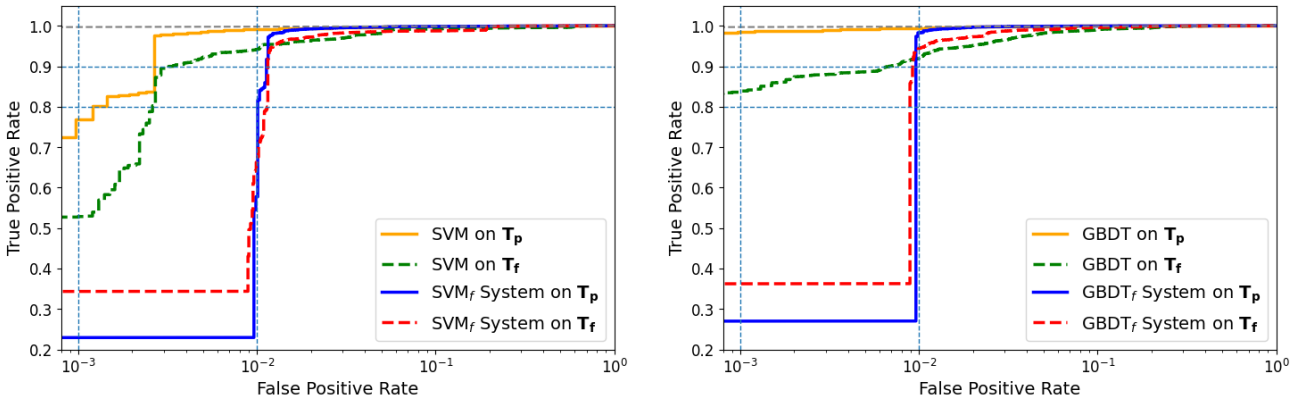
Evaluation Metrics. For performance and temporal analysis, we compute ROC curves, which show the True Positive Rate (TPR) and False Positive Rates (FPR) varying the decision threshold of the model. Regarding the robustness to adversarial EXEmples, we compute the Detection Rate (which is the TPR) at 1/% FPR as the manipulation size increases.

Hardware. To compute all our experiments, we leverage a workstation equipped with an Intel® Xeon(R) Gold 5420, two Nvidia L40 GPUs, and 540 GB of RAM.

3.2. Data Filtering

We train our models in two different configurations: (i) by filtering our training dataset with our YARA blocklist and allowlist thus training ML models only on data not recognized by signatures; and (ii) by using all the training data available, without filtering. Before we proceed towards training models though, we need to quantify the predictive capabilities in terms of TPR and FPR of the

10. docs.chocolatey.org



(a) (b)

Figure 3: ROC curves of all models and systems considered. We report SVM and SVM$_f$ System performances in Fig. 3a, while GBDT and GBDT$_f$ System performances are illustrated in Fig. 3b. Models trained on all data and AI systems that integrate YARA rules are tested with $\mathbf{T_p}$ (solid lines) and $\mathbf{T_f}$ (dashed lines).

rules we have collected, and we present this analysis in Tab. 1. The allowlist, which targets utilities of Windows, filters all the 2648 system files in the dataset as expected, while the blocklist removes 16329 malicious samples from the training data. At test time, the allowlist blocks 249 benign samples belonging to $\mathbf{T_p}$ and 0 samples belonging to $\mathbf{T_f}$. Instead, the blocklist blocks 1765 malicious samples of $\mathbf{T_p}$ and 6016 samples of $\mathbf{T_f}$. In other terms, the blocklist detect 24% of malware inside the Present Data ($\mathbf{D_t}$ and $\mathbf{T_p}$) and 34% of malware in Future Data ($\mathbf{T_f}$). For all datasets, we note that a non-negligible amount of FPR $\approx 10^{-2}$. By evaluating those through the responses of VirusTotal[11], we discover that they are caused by (i) *grayware programs* contained in the data we have described in Sec. 3.1, (ii) samples flagged by less than five AVs, or legitimate Python installers.

3.3. Performance and Temporal Analysis

We analyze the performance of the models we have trained, using both $\mathbf{T_p}$ and $\mathbf{T_f}$. We report in Fig. 3 the ROC curves of SVM and GBDT trained on all data, and we evaluate the same curves on corresponding AI systems trained on data filtered by the YARA rules. Regarding regularly-trained models, both SVM and GBDT achieve high TPR on $\mathbf{T_p}$ at 1% FPR, with SVM experiencing a consistent drop at extremely low FPR (0.1%). However, both models manifest a decay in performance when evaluated on $\mathbf{T_f}$ (green dashed line in Fig. 3). While GBDT confirms its capability at extremely low FPR, SVM maintains good performance only at 1%. As regards AI systems, ROC curves highlights the contribution of YARA rules, which guarantee a fixed TPR. However, they also force a fixed FPR, shown by the horizontal ROC until they reach almost 1% FPR. In fact, YARA rules report a fixed FPR of 0.96% on $\mathbf{T_p}$ and 0.89% on $\mathbf{T_f}$ as reported in Tab. 1. Beyond those points, the actual performance of SVM$_f$ and GBDT$_f$ becomes visible, showing similar or even improved results compared to models trained on the entire dataset. We see the AI system with GBDT$_f$ outperforming GBDT at 1% FPR on $\mathbf{T_f}$ while scoring almost the same on $\mathbf{T_p}$. The same holds for the AI system with SVM$_f$ compared to SVM, but at a slightly higher level of FPR. Lastly, we clarify why there is a mismatch between TPRs of AI Systems at very low FPR (left side of Fig. 3a and Fig. 3b). The AI System built with GBDT achieves higher TPR than the one built with SVM, and this difference is caused by some samples being detected by GBDT with full confidence, i.e. the output of the model is precisely 1. Hence, while computing the ROC it is not possible to fix a threshold for which those are misclassified as benign, thus causing the discrepancy with SVM, which does not exhibit this behavior.

Take-home Message 1: AI Systems can be trained on fewer data, matching the performance of models trained on all data. However, such performances are limited by the fixed FPR of YARA rules.

3.4. Robustness Analysis

The experimental analysis on robustness brings evidence of the contribution of YARA rules, coherently to what is presented by Ponte et al. [6]. We summarize all attack configurations using programs scraped from Chocolatey as content inject in Fig. 4a and Fig. 4b, and we report the results of attacks using Windows 10 programs in Fig. 4c and Fig. 4d. In general, as we expect, GAMMA finds more effective manipulations starting from out-of-distribution malware samples (dashed lines in all plots), since not only the TPR of models is lower in this condition, but also the attack is likely to produce samples deviating from the original training distribution. Also, the usage of Windows 10 benign sections results in smaller adversarial manipulations, since these programs are also smaller in size (on average) w.r.t. the Chocolatey ones, leading to lower evasion rates against SVM and GBDT, as shown in Fig. 4c and Fig. 4d. Instead, for SVM$_f$ and GBDT$_f$ systems, the trend holds for the first two points (10 and 20 section manipulations), but larger manipulations show a clear divergence from Fig. 4a and Fig. 4b. This

11. https://www.virustotal.com/gui/home/search

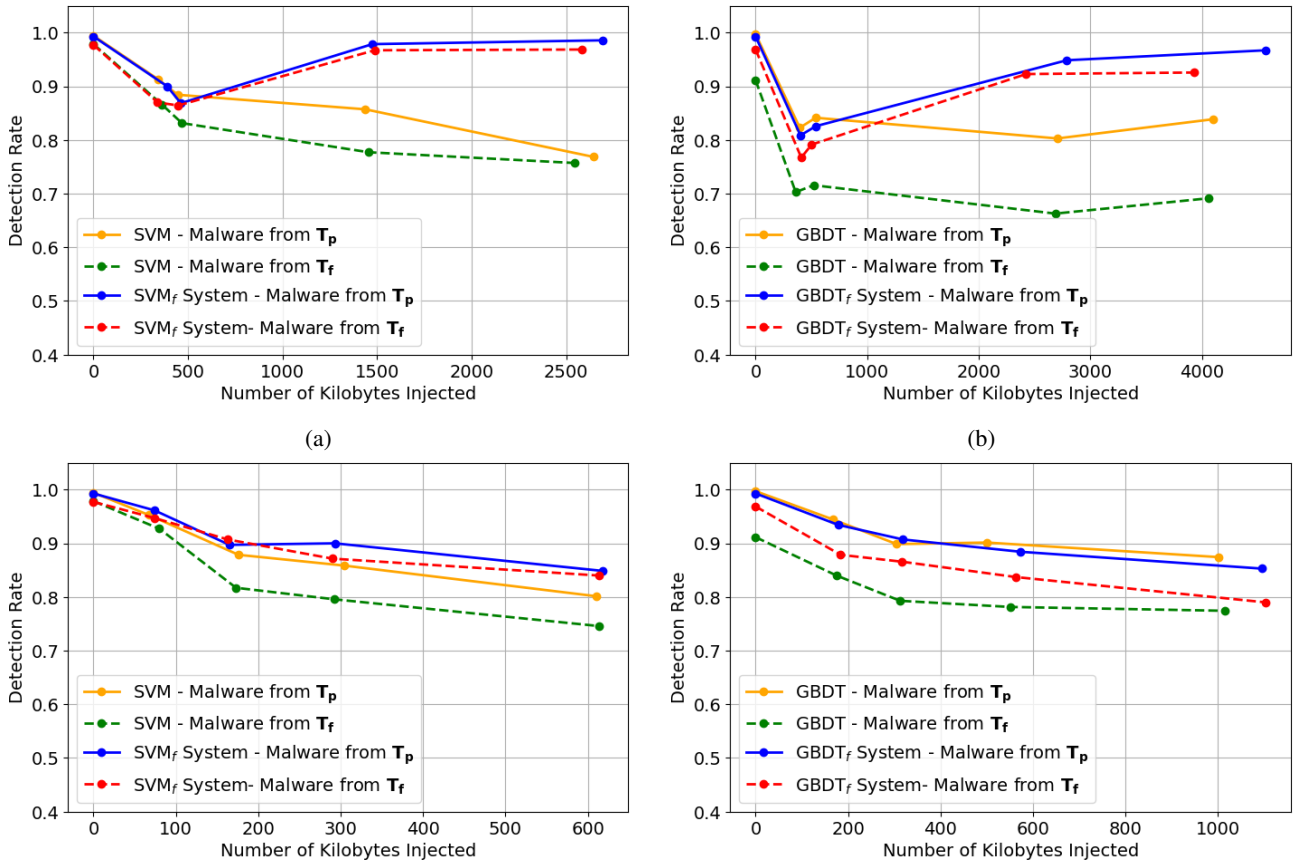


Figure 4: Detection rates of SVM, GBDT, SVM$_f$ System, and GBDT$_f$ System injecting Chocolatey (Fig. 4a and Fig. 4b) and Windows 10 (Fig. 4c and Fig. 4d) benign sections. Detection rates are reported based on the number of kilobytes injected (which varies according to *sections* parameter of GAMMA) and the source of malware samples. We report detection rates of adversarial EXEmples originated from $\mathbf{T_p}$ and $\mathbf{T_f}$ malicious samples with solid lines and dashed lines respectively.

phenomenon is caused mainly by two facts. The first reason is due to the inability of GAMMA of removing the patterns detected by rules from adversarial EXEmples. Hence, when untainted malware samples trigger a certain number of YARA rules, the corresponding adversarial EXEmple also triggers the same rules. The second reason is caused by GAMMA injecting particular strings that trigger rules, making the manipulation ineffective for evading the AI system. We clearly see this evidence in Fig. 4a and Fig. 4b: when injecting 10 and 20 sections the detection rate decreases or remains approximately the same, while when injecting 30 and 50 benign sections, this trend inverts. Examining the phenomenon in depth, we notice that a particular benign sample is included in 30 and 50 sections attacks, and when GAMMA crafts adversarial EXEmples using that goodware sections, `Typical_Malware_String_Transforms`[12] rule is always fired. We note that this YARA rule does not activate when tested on the original benign sample (which is a telemetry application), since it has been crafted to avoid such false positive. However, when GAMMA injects fragments of the sample, this condition is no longer met, leading to a significant number of attacks being blocked. We remark that, while previous work remarked that attackers should always remove false positives from their sources during content-injection attacks [6], the artifacts we detect in this work could not be foreseen in advance. That would have required the evaluation of each YARA rule by splitting their triggering conditions in all the single conditions chained with and/or operators, and re-evaluating them (and all their combinations) on benign samples owned by the attackers. This would introduce a combinatorial problem into the selection of legitimate programs to use for content-injection attacks, increasing the complexity of the attack setup. Lastly, our experimental analysis reports an unsteady trend of GBDT, which can be attributed to (i) a reduced set of considered sections (which in the original formulation of the attack is 75); and (ii) the randomness behind the genetic algorithm used to implement the attack itself. While SVM is characterized by a smooth decision function, easier to explore, GBDT models produce piece-wise constant decision regions that requires more random sampling performed by the optimization algorithm to find meaningful directions. Also, this can be seen in Fig. 4b, since the attack reduces the detection rate with a small amount of bytes, and then keeps roughly the same results for increasing strength of

12. https://github.com/Neo23x0/signature-base/blob/master/yara/gen_transformed_strings.yar

the attack. Finally, we observe that AI systems tend to be more robust than regular machine learning models, even when sections from Windows 10 utilities are injected. As previously mentioned, this is caused by the effectiveness of the blocklist, that can still effectively detect malware even after such manipulations.

Take-home Message 2: AI Systems are more robust to adversarial EXEmples due to the presence of rules. Also, content-injection attacks might include malicious patterns that trigger YARA rules, without being extracted by false positives of signature-based detection.

4. Conclusions

We now discuss the limitations of our methodology, by also highlighting the future lines of research that can take our work as a basis.

Limitations. While our analysis represents a step forward in understanding the properties of AI systems, we note the following limitations. First, the YARA rules used in this work were collected from public online repositories, and we did not filter them based on their precision on the training set. Although this filtering might have improved the performance of the AI systems, it could also have unfairly skewed the comparison in their favor. Moreover, our investigation would benefit from a broader model selection since we only considered one kernel for the SVM, and we did not fully explore the parameter space for the GBDT. Furthermore, we did not include complex deep neural architectures such as the one by Harang et al. [23]. We also tested the models' robustness only through content-injection attacks implemented by injecting sections with GAMMA, and did not consider other techniques proposed in the state of the art. In fact, other attacks might create different artifacts inside samples that might be detected by rules as well as the anomalies left by GAMMA. Nevertheless, we believe that the results showcased in Sec. 3 are valid, as we employed previously analyzed technologies that can be considered state of the art in this domain [9, 14, 15].

Future Work. While expanding our analysis to include more models and attacks is straightforward, we can further improve this investigation by exploring dynamic malware analysis, which collects the behavior of samples through execution in controlled environments (sandboxes). In particular, we could leverage signature-based detection in this context as well, for example by deploying CAPA[13] which identifies specific behavior patterns from sandbox reports. Moreover, regardless of the type of analysis, we could adapt the technique proposed by Scano et al. [24], which consists in training a linear model on the outputs of rules, thus weighting their answers to reduce false positives while maximizing the detection rate. Another improvement in our methodology could be using class weights at training time: tested models could benefit from this technique, possibly overcoming the problem of the unbalanced dataset.

13. https://github.com/mandiant/capa/

Final Remarks. In this work, we investigated AI Systems for malware detection, showing that these can match the performance of models trained on larger corpus of data while improving robustness, with a fixed amount of false alarms caused by rules themselves. Aligned with the upcoming recommendations from the EU Commission, we believe that extensive research should be conducted on developing and evaluating AI systems, moving away from the "in vitro" evaluation of isolated machine learning models. Hence, we believe that the preliminary results we collected on AI systems for malware detection could pave the way for further research, ultimately leading to the development of novel training and evaluation pipelines tailored to these technologies, which would also be compliant with emerging international guidelines and standards.

Acknowledgments

Andrea Ponte acknowledges the support of Rina Consulting S.p.A. for his doctoral scholarship and research work. The authors acknowledge Matous Kozak's help with data collection. This work was partially supported by projects SERICS (PE00000014) and FAIR (PE00000013) under the NRRP MUR program funded by the EU - NGEU.

References

[1] European Commission. European ai act. https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai, 2024. Accessed: February 2025.

[2] ESET Technology. The multilayered approach and its effectiveness. https://www.eset.com/fileadmin/ESET/US/docs/about/ESET-Technology-Whitepaper.pdf, 2017. Accessed: February 2025.

[3] Avira. Nightvision – using machine learning to defeat malware. https://www.webassetscdn.com/avira/prod/cache-buster-1598423379/assets/oem.avira.com/resources/to%20delete/whitepaper_NightVision_EN_20170704.pdf, 2017. Accessed: February 2025.

[4] Microsoft. Evolution of malware prevention. https://info.microsoft.com/rs/157-GQE-382/images/Windows%2010%20Security%20Whitepaper.pdf, 2017. Accessed: February 2025.

[5] Apple. Protecting against malware in MacOS. https://support.apple.com/guide/security/protecting-against-malware-sec469d47bd8/web. Accessed: April 2025.

[6] A. Ponte, D. Trizna, L. Demetrio, B. Biggio, I. T. Ogbu, and F. Roli. Slifer: Investigating performance and robustness of malware detection pipelines. *Computers & Security*, 150:104264, 2025.

[7] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *ITASEC CEUR Workshop Proceedings*, 2019.

[8] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli. Adversarial EXEmples: A survey and experimental evaluation of practical

attacks on machine learning for windows malware detection. *ACM Transaction on Privacy and Security*, 24(4):1–31, 2021.

[9] H. S. Anderson and P. Roth. Ember: an open dataset for training static pe malware machine learning models. *Preprint, arXiv:1804.04637*, 2018.

[10] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *International Conference on Malicious and Unwanted Software*, 2015.

[11] E. Raff, J. Barker, J. Sylvester, R. Brandon, . Catanzaro, and C. K. Nicholas. Malware detection by eating a whole exe. In *AAAI Conference on Artificial Intelligence*, 2018.

[12] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. Malware images: visualization and automatic classification. In *International Symposyum on Visualization for Cyber Security*, 2011.

[13] L. Liu, B. S. Wang, B. Yu, and Q. X. Zhong. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, 18(9):1336–1347, 2017.

[14] D. Trizna. Quo vadis: hybrid machine learning meta-model based on contextual and behavioral malware representations. In *ACM Workshop on Artificial Intelligence and Security*, 2022.

[15] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transaction on Information Forensics and Security*, 16:3469–3478, 2021.

[16] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022.

[17] N. Cristianini and E. Ricci. Support vector machines. In *Encyclopedia of algorithms*. Springer-Verlag, 2008.

[18] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, and Others. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[21] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *ACM International Conference on Knowledge Discovery and Data Mining*, 2016.

[22] L. Oneto. *Model selection and error estimation in a nutshell*. Springer, 2020.

[23] R. Harang and E. M. Rudd. SOREL-20M: A large scale benchmark dataset for malicious PE detection. *Preprint, arXiv:2012.07634*, 2020.

[24] C. Scano, G. Floris, B. Montaruli, L. Demetrio, A. Valenza, L. Compagna, D. Ariu, L. Piras, D. Balzarotti, and B. Biggio. Modsec-learn: Boosting modsecurity with machine learning. In *International Conference on Distributed Computing and AI*, 2024.