ObfusGate: Representation Learning-Based Gatekeeper for Hardware-Level Obfuscated Malware Detection

Zhangying He, Chelsea William Fernandes, and Hossein Sayadi
Department of Computer Engineering and Computer Science, California State University, Long Beach, CA, USA

***Abstract*—In this paper, we explore the interplay between code obfuscation techniques and performance counter traces to undermine Hardware Malware Detectors (HMDs) that rely on Machine Learning (ML) models. By crafting various obfuscated malware categories and analyzing a wide range of ML models, we demonstrate a notable detection performance reduction, showcasing the evasive impact of obfuscated malware in HMD methods. To counter these threats, we propose *ObfusGate*, an intelligent and robust defense mechanism based on feature representation learning that significantly enhances machine learning models against both obfuscated and unobfuscated malware attacks. The results indicate the effectiveness of *ObfusGate*, attaining up to 24% detection rate increase across diverse ML models assessed for hardware-level obfuscated malware detection at run-time.**

***Index Terms*—Code Obfuscation, Cybersecurity, Hardware Performance Counters, Machine Learning, Malware Detection.**

I. Introduction and Background

Emerging cybersecurity challenges have led to advanced defense strategies combining traditional signature-based methods with machine learning-based detectors. However, attackers employing diverse obfuscation methods present a challenge to these defenses. The inefficiencies in software-based techniques have spurred the development of Hardware Malware Detection (HMD) [1]–[3]. Utilizing microarchitectural features from Hardware Performance Counters (HPCs) alongside ML, HMDs excel in distinguishing between malicious and legitimate programs, surpassing traditional software-based solutions.

Previous HMD methods have overlooked the intricate challenge presented by malware obfuscation techniques, ignoring their evasion potential and the need for specialized malware detection mechanisms. Meanwhile, prior research on obfuscated malware detection mainly concentrates on generic malware identification in Android apps [4]–[6] and IoT side-channel obfuscation attack [7], with limited hardware-level exploration. These studies often focus narrowly on specific malware obfuscation types, neglecting hackers' use of varied obfuscation methods combined with regular malware.

In response, in this work we develop various code obfuscation techniques within HMD systems, revealing their ability to circumvent ML-based hardware malware detectors via malware obfuscation. Subsequently, we introduce *ObfusGate*, an intelligent and robust defense mechanism based on feature representation learning specialized in securing systems against blended attacks, including diverse obfuscated and unobfuscated malware. The considered threat model in *ObfusGate* addresses the real-world threat landscape encountered by contemporary security systems. Furthermore, within *ObfusGate*, our investigation assesses the viability of employing a unified defense model for various malware types. We integrate a feature representation learner uniquely trained on unobfuscated malware HPC data which is capable of effectively discerning significant malicious patterns from noises embodied in obfuscated malware's HPCs. This approach demonstrates consistent effectiveness in strengthening ML-based malware detectors, showcasing adaptability and versatility across various real-world attacks.

II. Proposed Methodology

a) Framework Overview: Fig. 1 depicts the integrated deployment of *ObfusGate*, involving hardware-level attacks monitoring, feature engineering, representation learning, and ML defense. This comprehensive defense mechanism effectively handles both obfuscated and regular malware attacks. An extensive range of hardware-level features are collected from both benign and malware applications using the Perf tool on an Intel Core i7 processor running Ubuntu 22.04.2. Dataset is divided into an 80:20 ratio for training and testing ML models.

b) Malware Obfuscation: We analyzed the robustness of *ObfusGate* against four common code obfuscation techniques: null code insertion, code splitting, code shuffling, and variable substitution. Null code insertion adds non-functional code blocks, complicating malware code. Code splitting fragments the code into modules across multiple files, impeding code analysis. Code shuffling rearranges code structure to obscure the program's logic. Variable substitution replaces variable names with meaningless ones, impeding automated analysis. These techniques increase the code complexity, hindering reverse engineering. We rigorously analyzed their effectiveness in circumventing ML models within HMD methods.

c) Representation Learning: The system first operates with ML models trained on regular malware attacks. As attack scenarios diversify due to various obfuscation techniques, *ObfusGate* adapts by incorporating a new training and inference process. It employs a denoising model, a neural network using a denoising autoencoder. This autoencoder is trained on unobfuscated malware features intentionally augmented with noise, enabling the model to learn a representation that maps corrupted data back to its original, uncorrupted state in an unsupervised manner. During inference, this model reconstructs vital features, filtering out noise from inputs affected by obfuscated malware. Unlike prevalent methods training on obfuscated malware data, *ObfusGate* uniquely relies on unobfuscated malware features, using known patterns to counter evolving obfuscation techniques.

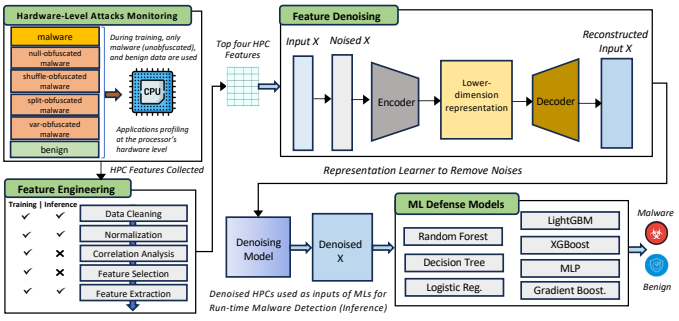

Fig. 1: Overview of *ObfusGate* for detecting obfuscated malware at run-time.

979-8-3503-4859-0/DATE24/© 2024 EDAA

TABLE I: 1) Performance (F1-score) and downgrade (%) of obfuscated malware detection, 2) Performance (F1-score) and boost-up rate (%) of *ObfusGate*, with overhead results.

ML-based Detector	1) Unobfuscated & Obfuscated Malware Attacks							2) *ObfusGate*: Unobfuscated & Obfuscated Malware Defense											
	F1-score					latency	memory	F1-score					Boost-up Rate					latency	memory
	unobf	null	shuf	split	var	(ms)	(KB)	unobf	null	shuf	split	var	unobf ↑	null ↑	shuf ↑	split ↑	var ↑	(ms)	(KB)
Random Forest	0.93	0.81	0.82	0.79	0.84	0.037	606	0.96	0.92	0.91	0.91	0.92	3%	11%	9%	12%	8%	0.078	1,742
Decision Tree	0.9	0.78	0.8	0.75	0.81	0.002	9	0.96	0.90	0.86	0.88	0.91	6%	12%	6%	13%	10%	0.048	158
Logistic Regression	0.88	0.75	0.76	0.67	0.74	0.003	1	0.94	0.92	0.91	0.90	0.91	6%	17%	15%	23%	17%	0.053	144
MLP	0.84	0.7	0.7	0.68	0.67	0.006	1,050	0.95	0.91	0.91	0.91	0.91	11%	21%	21%	23%	24%	0.058	1,742
XGBoost	0.94	0.81	0.83	0.79	0.84	0.010	162	0.96	0.92	0.92	0.90	0.91	2%	11%	9%	11%	7%	0.057	300
LightGBM	0.94	0.81	0.83	0.8	0.84	0.006	327	0.96	0.91	0.92	0.91	0.91	2%	10%	9%	11%	7%	0.063	501
Gradient Boosting	0.93	0.8	0.83	0.77	0.84	0.003	163	0.96	0.91	0.91	0.90	0.91	3%	11%	8%	13%	17%	0.050	307

d) Training Phase: Exclusively relying on unobfuscated input for training, we introduced Gaussian noise to the training data. This corrupted data was utilized to train the denoising model, enabling it to encode vital feature representations and reconstruct the original, uncorrupted input. Subsequently, these reconstructed features were employed to train the ML models, which were then deployed during the inference phase.

e) Inference Phase: As shown in Fig. 1, the system processes diverse attack scenarios. HPC features undergo monitoring and preprocessing before entering the denoising model. This model reconstructs the feature space by preserving crucial features (clean features) while eliminating noise from obfuscated data (and noise from unobfuscated input in regular malware attacks). The refined features then feed into various range of ML models trained on unobfuscated data, enabling accurate and robust malware detection, adaptable to different attack scenarios.

III. Experimental Results and Evaluation

a) Obfuscated Malware Attacks: Table I (part 1) showcases the results of top ML models for unobfuscated and four types of obfuscated malware attacks (in red). The results clearly show how all four obfuscation techniques diminish the system's F1-score, indicating their success in bypassing the ML defense model. Split obfuscation notably undermines system security by an average of 16% (max 21%). Despite XGBoost and LightGBM exhibiting strong defense (both at 94% F1-score), obfuscated attacks notably reduce their True Positive Rates (TPR) (not shown due to space limitation) from 89.4% to 66.7% for XGBoost and from 88.9% to 67.6% for LightGBM.

b) Defending against Obfuscated Malware Attacks: Table I (part 2) presents *ObfusGate*'s F1-scores (in green) and boost-up rates for leading ML models. On average, *ObfusGate* increases the detection rate against all four types of obfuscated attacks by 13%. Notably, for the highly concealed split obfuscation, it elevates all ML models' average F1-scores by 16%. It further augments LightGBM and XGBoost, elevating their F1-scores from 80% to 91% and 79% to 90%, respectively. A key aspect is *ObfusGate*'s consistent performance across unobfuscated and various obfuscated attacks, allowing one ML model deployment for defending diverse attack scenarios. Moreover, it improves defense against regular malware attacks by an average of 5%. Remarkably, MLP improved by 11% in F1-score for regular malware and 23% for split obfuscated attacks detection.

c) Efficiency Trade-off Analysis: Fig. 2 illustrates the detection rate (F1-score) vs. overhead (latency*memory footprint) efficiency analysis before and after *ObfusGate* against split obfuscated malware attacks during run-time inference. The inference latency and memory overhead results are reported in Table I. Logistic Regression (LR), XGBoost, LightGBM, and Gradient Boosting (GB) distinguish themselves as efficient defenders. LR model proves most cost-efficient with 0.05ms inference time and 144KB model size while offering a significant detection rate. Additionally, XGBoost and LightGBM show stability and speed with relatively compact models.

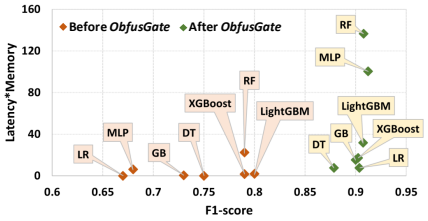

Fig. 2: Detection Rate (F1-score) vs. Cost Efficiency (Latency*Memory Footprint) analysis for the most effective obfuscated malware attack (split obfuscation).

d) Adaptability Analysis across Attack Scenarios: To assess *ObfusGate*'s adaptability to evolving attack scenarios, we designed six distinct attacks encompassing unobfuscated malware, split, null, shuffle, and var attacks. Table II presents the results for XGBoost, one of the top-performing models. *ObfusGate* consistently achieves an F1-score and accuracy of 90% or higher in all attack scenarios. Moreover, in detecting unobfuscated malware attacks, it obtains a 100% TPR and recall, showcasing its effectiveness in adapting to diverse malware attacks.

TABLE II: XGBoost's detection rate (F1-score) in *ObfusGate* across various attacks.

Scenario	Accuracy	F1-score	AUC	TPR	FPR	FNR	TNR	Precision	Recall
attack1	0.95	0.96	0.96	1.00	0.09	0.00	0.91	0.92	1.00
attack2	0.90	0.90	0.93	0.89	0.09	0.11	0.91	0.91	0.89
attach3	0.93	0.93	0.95	0.95	0.09	0.05	0.91	0.92	0.95
attack4	0.93	0.93	0.94	0.94	0.09	0.06	0.91	0.91	0.94
attack5	0.92	0.92	0.94	0.94	0.10	0.06	0.90	0.90	0.94
attack6	0.92	0.92	0.94	0.94	0.10	0.06	0.90	0.90	0.94

attack1: unobfuscated malware attack; **attack2**: split attack; **attack3**: unobfuscated and split malware attacks; **attack4**: unobfuscated, split, and null attacks; **attack5**: unobfuscated, split, null, and shuf attacks; **attack6**: unobfuscated, split, null, shuf, and var attacks.

IV. Conclusion

In this work, we demonstrated the efficacy of code obfuscation in circumventing machine learning-based hardware malware detectors, leading to a significant decrease of (9-21)% in detection rate across different models. We further introduced *ObfusGate*, a representation learning-based gatekeeper to boost up the performance of detectors against both obfuscated and unobfuscated malware attacks. *ObfusGate* focused on distinguishing crucial patterns of malicious behavior from noise, reducing reliance on the unpredictability of obfuscated malware.

V. Acknowledgment

This work is supported by the National Science Foundation under Award No. 2139034.

References

[1] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *ISCA*. ACM, 2013, pp. 559–570.

[2] M. Ozsoy *et al.*, "Malware-aware processors: A framework for efficient online malware detection," in *HPCA*, 2015, pp. 651–661.

[3] H. Sayadi *et al.*, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Design Automation Conference (DAC)*, 2018, pp. 1–6.

[4] L. Onwuzurike *et al.*, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Trans. Priv. Secur.*, vol. 22, no. 2, apr 2019.

[5] M. Ikram *et al.*, "Dadidroid: An obfuscation resilient tool for detecting android malware via weighted directed call graph modelling," *arXiv*, 2019.

[6] Z. Li *et al.*, "Obfusifier: Obfuscation-resistant android malware detection system," in *SecureComm*. Springer, 2019, pp. 214–234.

[7] D.-P. Pham *et al.*, "Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification," in *ACSAC'21*, pp. 706–719.