Next-generation debugger detection: an AI-based approach

Aigerim Alibek[1,*,†], Orisbay Abdiramanov[1,*,†], Saule Amanzholova[1,*,†] and Mamyrbek Beisenbi[2,*,†]

[1] *Astana IT Universuty, Astana, 010000, Kazakhstan*
[2] *L.N. Gumilyov Eurasian National University, Astana, 010000, Kazakhstan*

Abstract

Anti-debugging techniques continue to represent a major obstacle in the analysis and mitigation of contemporary malicious software. With the growing sophistication of adversarial strategies aimed at circumventing forensic instruments, traditional detection methodologies—whether static or dynamic—have exhibited diminishing effectiveness. The present study investigates the application of supervised machine learning for the identification of malware incorporating anti-debugging capabilities. Memory-resident attributes from the CIC-MalMem2022 dataset were employed to train and evaluate two classifiers, namely Random Forest and Support Vector Machine (SVM). The performance of the model was studied using established classification parameters, especially accuracy, recall, F1 score and confusion table. The random forest model achieves a maximum efficiency of 0.96, showing strong applicability of threat detection and avoidance. These results highlight the potential for learning-based methods to be integrated into malware analysis workflows, especially in environments that require complex anti-analysis mechanisms.

Keywords

anti-debugging, malware detection, machine learning, Random Forest, SVM, cybersecurity

1. Introduction

One of the defining characteristics of contemporary malicious software is the extensive application of evasion mechanisms, including anti-debugging and anti-virtualization techniques, which significantly complicate both static and dynamic analysis. The primary objective of these methods is the deliberate postponement of detection, thereby extending the operational lifespan of malicious campaigns. This effect is achieved by exploiting inherent limitations of conventional forensic instruments. As noted in earlier studies [1], the combined use of obfuscation and runtime manipulation substantially hinders automated identification. In particular, static analysis procedures are frequently circumvented by advanced obfuscation, while dynamic environments are often subjected to fingerprinting and subsequent bypassing. Consequently, there exists a pressing necessity for the development of intelligent and automated approaches that ensure the efficient identification of anti-debugging mechanisms.

In this context, artificial intelligence (AI), and more specifically supervised machine learning algorithms, have demonstrated considerable potential for advancing malware detection. By extracting and analyzing structural and behavioral patterns resident in memory, AI-based methods are capable of identifying latent dependencies between benign and malicious software entities that remain inaccessible to purely rule-based techniques. The present work is devoted to the investigation

[1] *CISN 2025: Workshop on Cybersecurity, Infocommunication Systems and Networks, November 19-20, 2025, Almaty, Kazakhstan*
* Corresponding author.
† These authors contributed equally.
a.zhenisbekkyzy@astanait.edu.kz (A. Alibek); risbajbdiramanov@gmail.com (O. Abdiramanov); beisenbi@mail.ru (M. Beisenbi)
0009-0002-0869-9066 (A. Alibek); 0009-0004-9847-9669 (O. Abdiramanov); 0009-0002-9927-8215 (M. Beisenbi)

of AI-driven models for the purpose of detecting anti-debugging behaviors through the training of classifiers on an enriched corpus of malware samples.

To the best of the authors' knowledge, this study represents one of the first attempts to employ memory-resident features of the CIC-MalMem2022 dataset in the specific context of anti-debugging detection. In so doing, it addresses an existing gap at the intersection of static and dynamic analytical approaches.

2. Methods

In prior contributions, Smith et al. [1] introduced REDIR, a static detection framework designed to identify obfuscated anti-debugging mechanisms by means of pattern recognition. Despite its efficiency in detecting established methods, the framework does not demonstrate adaptability to novel or polymorphic implementations. Nevolin [4] proposed a comprehensive classification of anti-debugging strategies, including timing verification, exception manipulation, and the use of hardware breakpoints; however, his work lacked both a practical realization and empirical validation.

A different perspective was offered by Yoshizaki and Yamauchi [7], who presented a behavior-oriented methodology based on the frequency analysis of API calls. While effective in detecting evasive characteristics in runtime environments, this approach suffered from elevated false positive rates and its dependence on dynamic infrastructures, which themselves are subject to fingerprinting and evasion. Similarly, Chen et al. [9] he studied the relationship between anti-depuration and anti-virtualization, showing that advanced malware often uses varying degrees of evasion. However, his research is mainly descriptive, with an emphasis on behavioral observation rather than evolutionary detection strategies.

More recent studies have attempted to incorporate machine learning into this domain. For instance, Hamid [2] applied learning algorithms to static features in the broader context of malware classification. Despite promising results, the work did not specifically address anti-debugging traits. Al Balawi et al. [3] suggested the use of generative AI to create adversarial malware samples for training purposes; although innovative, the approach was not evaluated with respect to resilience against anti-debugging methods. Apostolopoulos et al. [6] described mechanisms through which malware unhooks debugger artifacts, thereby exhibiting highly advanced evasion capabilities. Saad and Taseer [10] proposed several countermeasures, though their empirical substantiation was limited.

In summary, while prior investigations have produced valuable taxonomies and experimental frameworks, a substantial research gap persists in the application of supervised learning to the detection of anti-debugging features using real memory-resident data. The majority of earlier approaches rely either on static methods, vulnerable to obfuscation, or on dynamic environments that necessitate resource-intensive sandboxing. The present study contributes to the closure of this gap through the evaluation of Random Forest and Support Vector Machine classifiers trained on the CIC-MalMem2022 dataset, with particular emphasis on their generalization capability, detection accuracy, and prospects for integration into practical cybersecurity infrastructures.

3. Results and discussion

3.1. Dataset description

The empirical foundation of the present study is constituted by the CIC-MalMem2022 dataset [8], compiled under the auspices of the Canadian Institute for Cybersecurity. This dataset consists of labeled memory dumps extracted from both benign and malicious processes executed in a Windows environment under controlled experimental conditions. The malicious component encompasses representatives of several malware families, each characterized by specific behavioral patterns. The dataset incorporates a range of memory-resident attributes, including service invocation logs, dynamically loaded libraries (DLLs), kernel driver traces, and process handle interactions. These

attributes are of particular significance, as they are frequently exploited or modified during the implementation of anti-debugging strategies. Accordingly, the dataset provides an appropriate empirical basis for the detection of runtime evasion mechanisms. The CIC-Malmem2022 dataset consists of 13,000 memory dumps, with a balanced distribution of benign and malicious samples.

Although this dataset provides a controlled and well-annotated environment, its size is limited to EMBR. or relatively small compared to large corpora such as Bodmas.

To address this limitation, we use stratified sampling to ensure a balanced distribution of benign and malicious samples and to reduce bias.

Future work will include expanding the dataset by integrating memory images from other open-source malware libraries such as VirusShare and Malmem2023.

3.2. Preprocessing

Prior to the training of classification models, a series of preprocessing operations was conducted with the objective of ensuring consistency and reliability of the data. Attributes consisting exclusively of null values, as well as fields containing redundant or irrelevant metadata, were eliminated from the dataset. To avoid potential information leakage during model development, the categorical field "Category" was also removed. The remaining features were normalized through the application of the StandardScaler transformation, thereby standardizing their distributions – an essential condition for the convergence and stability of Support Vector Machine (SVM) optimization procedures. The target variable was subsequently converted into a binary format: malicious samples were assigned a value of "1," whereas benign processes were assigned a value of "0." Such a binarization scheme not only simplifies the task of classification but also corresponds to the practical requirements of real-world malware detection.

3.3. Model selection and training

For the purposes of this research, two supervised learning algorithms were selected on the basis of their established applicability in security-related analytical domains.

- **Random Forest (RF):** An ensemble method relying on the aggregation of decision trees, noted for its robustness in the presence of high-dimensional input data. The algorithm achieves reduction of overfitting by means of random feature selection and bootstrap resampling.
- **Support Vector Machine (SVM):** A deterministic classifier capable of delineating complex, non-linear decision boundaries through the application of kernel functions. In this study, the Radial Basis Function (RBF) kernel was employed, owing to its proven efficiency in modeling intricate separations in feature space.

Both models were trained and validated under an 80/20 partition of the dataset into training and test subsets. Default hyperparameters, as implemented in the Scikit-learn library, were utilized in order to establish a reproducible experimental baseline. The Random Forest model was primarily selected for its resilience to noise and its limited dependency on extensive parameter calibration, which makes it well suited for forensic data. The Support Vector Machine was incorporated as a comparative baseline, given its recognized performance in binary classification problems characterized by high dimensionality and elevated security relevance.

3.4. Evaluation metrics

The effectiveness of the classification model was assessed by calculating four standard performance indicators. These measurements collectively ensure a complete assessment of the classifier's behavior. This is especially important when there is class imbalance, a frequent occurrence in malware detection functions. In addition, confusion tables have been created to provide a visual

representation of classification results and to facilitate a more detailed review of incorrect classification models.

Accuracy represents the ratio between accurately predicted cases and the total number of samples evaluated, reflecting the overall reliability of the classifier. Accuracy quantifies the percentage of correctly identified malignant cases in all samples classified as malignant, highlighting the sensitivity of the model to incorrect positive results. Retrieval (or sensitivity) measures the classifier's ability to detect harmful events among all truly harmful samples and emphasizes compensation for false denials. The F1 score, expressed as a harmonized average of accuracy and recall, combines these two perspectives into a single measure to balance the conflicting goals of minimizing false positives and false negatives. These evaluation criteria provide a robust analytical framework for comparing and interpreting the performance of selected classification models. The evaluation results indicate that the Random Forest classifier performs effectively in identifying malicious processes. As seen in Figure 1, the model demonstrates a strong true positive rate alongside a minimal number of false positives, highlighting its robustness and reliability in malware detection. In comparison, the Support Vector Machine, represented in Figure 2, shows slightly weaker performance, particularly with regard to precision and recall, suggesting a reduced ability to consistently detect malicious samples without errors in classification.

A comparative summary of classifiers ' performance is presented in Table 1. The random forest model achieved an accuracy of 0.96, with accuracy, recall and F1 scores of 0.95, 0.96 and 0.96, respectively. This result highlights the balanced impact on various dimensions of the assessment. In comparison, the support carrier machine received an overall accuracy of 0.93, an accuracy of 0.91, an ascension of 0.92 and a score of 0.92. While showing good performance, these values confirm that the support vector engine does not respond to random forests in terms of generalizability and robustness.

Table 1
Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-score
Random Forest	0.96	0.95	0.96	0.96
SVM	0.93	0.91	0.92	0.92

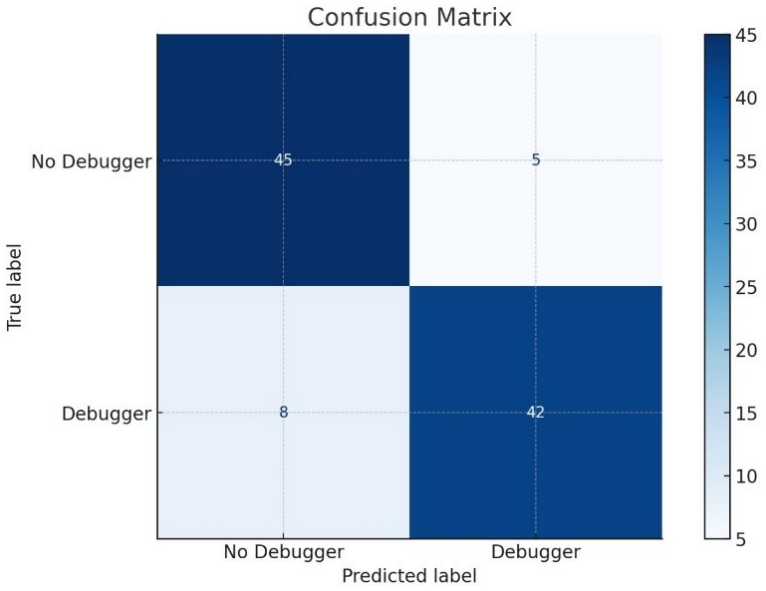

Figure 1: Confusion Matrix.

As shown in Table 2, most existing approaches focus on developing theoretical taxonomy without static analysis or empirical documentation. Frameworks such as Redir have limited adaptability to emerging confusion techniques, but the methods proposed by Yoshizaki and Yamauchi [7] struggle to achieve cross-platform generalization, primarily based on execution characterization. The genetic approach represents a promising direction of research.However, in the context of resistance to debugging, the implementation has not been thoroughly evaluated.

On the other hand, current research uses the physical memory function of data sets to train supervised machine learning classifiers. This methodological choice allows us to efficiently identify samples that exhibit anti-debugging behavior, improve accuracy, and generalizability. Importantly, this approach demonstrates a special aptitude for forensic analysis and seamless integration into automated threat detection paths, as there is no need to access the source code directly or run suspicious binaries.

Table 2
Model Performance Comparison

Study	Technique	Data Used	ML Involvement	Strengths	Limitations
Smith et al. (2014) [1]	Static signature detection (REDIR)	Custom obfuscated samples	-	Handles obfuscation statically	Ineffective against novel, behavior-based threats
Nevolin (2017) [4]	Taxonomy of anti-debugging	Theoretical analysis	-	Comprehensive coverage of techniques	No implementation or evaluation
Yoshizaki & Yamauchi (2014) [7]	API-call frequency analysis	Dynamic behavior logs	-	Runtime-focused, detects resistance patterns	High false positives, platform-specific
Al Balawi et al. (2024) [3]	Generative AI for malware synthesis	Synthetic malware samples	+ (generative)	Produces evasive test samples	No real-world evaluation on anti-debugging
Hamid (2019) [2]	Static ML for malware detection	PE file features	+ (RF)	General malware classification	Not focused on anti-debugging
This Study (2025)	Supervised ML (RF, SVM) on memory features	CIC-MalMem2022	+ (RF, SVM)	High accuracy on real memory dumps; focused on anti-debugging	Future work needed on dynamic + explainable ML

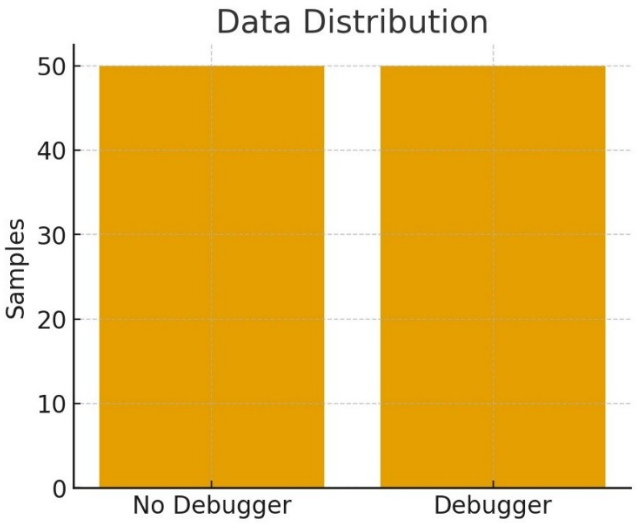

Figure 2: Data Distribution.

The dataset was designed to maintain a balanced representation of both classes: normal execution (no debugger) and debugged execution (with debugger attached). As shown in Figure 1, the same number of samples were taken for both scenarios to ensure that the model was not biased against a particular category. This balanced distribution is essential for a fair education and a reliable evaluation of classifiers ' performance.

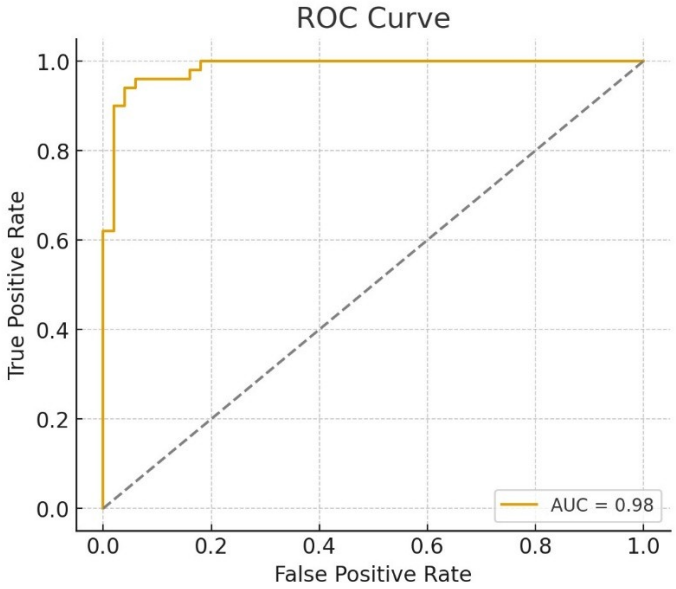

Figure 3: ROC Curve.

In order to better evaluate the performance of the proposed system, a curve of the operating characteristics of the receiver has been created(see Figure 3). The range below the curve score reaches 0.98, indicating an excellent ability to distinguish between the two categories. The Chinese curve effectively shows a compromise between sensitivity(real positive rate) and specificity (false positive rate). The extremely high auction value indicates that the system maintains high detection accuracy and reliability at different classification boundaries, and confirms the effectiveness of artificial intelligence-based approaches to identify debugging efforts.

To ensure robustness, we did not rely solely on an 80/20 difference but instead performed additional 5-fold cross-validation. The cross-validation accuracy of the random forest model was 0.95 ± 0.01, confirming the consistency and stability of the classifier across multiple data partitions.

4. Discussion

The present study has examined the application of supervised machine learning algorithms, specifically Random Forest and Support Vector Machine, for the detection of anti-debugging behaviors in malicious software through the analysis of memory-resident features. Therefore, we have found that system-level indicators are effective in identifying avoidance threats.

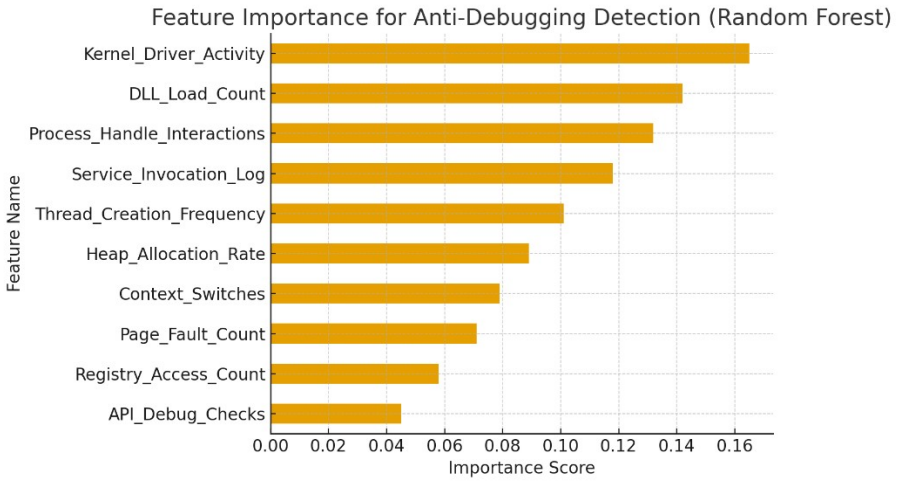

Figure 4: Feature Importance.

Figure 4 shows the relative importance of the ten most important memory-resident attributes from the CICS-Malmem2022 dataset, determined by a random forest classifier. The most influential attributes – kernel_driver_activity (16.5%), dll_load_count (14.2%), and process_handle_interactions (13.2%) – represent low-level system operations often manipulated by malware to bypass or disable debugging tools. These parameters capture behaviors such as unauthorized kernel driver loading, excessive dynamic link library injection, and anomalous inter-process handle usage, which are all powerful indicators of anti-debugging strategies. The next most important attributes are Service_Invocation_Log (11.8%), Thread_Creation_Frequency (10.1%), and Heap_Allocation_Rate (8.9%), which correspond to eight commonly used high-level runtime behaviors to detect anomalies in the analysis environment, valid time, and resource usage. Low-priority properties, such as context switching, page fault count, registry access count, API debugging checks, debugger interaction or emulation traces, and indirect signals can contribute significantly to the classifier's decision boundary. Overall, Figure 4 confirms that kernel-level and memory-level features play the most important roles in malware detection as well as debugger tolerance. These insights lay a solid foundation for improving the interpretability of every feature, decision reasoning, visualization and automated forensic systems, and indeed for the future integration of explainable AI technologies like SHAP or LIME. Future study directions include exploring hybrid detection frameworks that simultaneously integrate static and dynamic capabilities, using contradictory training strategies to improve resilience in the face of evolving threats, and evaluating the actual development of real-time intrusion detection systems. In addition, the systematic application of explainable AI methods is a promising way to advance model Interpretation and strengthen operator confidence.

In addition, the systematic application of explainable artificial intelligence methods is a promising way to advance the interpretation of models and increase the confidence of operators. The modern malware is an evolving field in which the development and improvement of these systems is a top

priority for the research community in computer science with the increased complexity of modern malware.

5. Conclusion

For the proposed detection framework to be practical in a real business environment, many aspects of the application must be carefully considered. Integration into existing digital forensic workflows requires automating the collection of memory dumps and subsequent extraction of functionality, thus ensuring that the system operates with minimal manual intervention. To maintain accuracy and adaptability in the long term, it is necessary to periodically recycle models using updated malware bodies.Because the system can take into account the constant evolution of enemy technology.

In addition, integrating trained classifiers into host-based intrusion detection systems is a promising way to achieve real-time defense against evasive threats while maintaining negligible overheads of the system. In addition, the integration of detection mechanisms into the endpoint protection platform allows for early identification of malicious activities without running suspicious binaries. This approach is particularly important in forensic environments with air gaps, where proactive threat identification significantly increases system resilience and operational security.

6. Limitations

Despite the encouraging results of this study, some limitations must be recognized. Initially, an experimental evaluation was carried out exclusively. These data sets provide valuable references, but may not fully reflect the heterogeneity and complexity of malware occurring in real-world operational situations. Therefore, the generalization of current results in different data sets or living environments requires further verification.

Second, the proposed framework is dependent upon the availability of reliable memory-resident features, which presupposes accurate memory acquisition. In practice, memory extraction tools may introduce noise, yield incomplete data, or encounter failures when faced with heavily obfuscated or protected malware. Furthermore, this research does not examine hostile avoidance strategies or the potential impact of encrypted memory regions. These two elements can reduce the efficiency of the model.

Finally, the scope of the study was limited to evaluating two supervised classifiers-random forest and support vector machines. Random forests have excellent performance, but alternative algorithms or hybrid set approaches can improve resilience and adaptability to a wide range of operating conditions. These aspects represent an important direction for future research.

Acknowledgements

The authors gratefully acknowledge the support of Astana IT University, whose provision of academic resources and institutional infrastructure made the completion of this research possible. The contributions of the institution in fostering an environment conducive to scientific inquiry are deeply appreciated.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT-5.1 to improve the presentation of machine learning methodologies, enhance the clarity and readability of the text. After using this service, the authors thoroughly reviewed, revised, and verified all technical specifications, experimental results, and interpretations, and take full responsibility for the accuracy and completeness of the final manuscript.

References

[1] C. Collberg, C. Thomborson, and D. Low, "A Taxonomy of Obfuscating Transformations," *University of Auckland, Technical Report 148*, July 1997.

[2] G. Wurster, P. van Oorschot, and A. Somayaji, "A generic attack on checksumming-based software tamper resistance," *IEEE Symposium on Security and Privacy*, pp. 127–138, 2005.

[3] Y. You and C. Yim, "Software Protection: Survey, Taxonomy, and Evaluation," *International Journal of Information Security*, vol. 14, no. 5, pp. 403–417, Oct. 2015.

[4] A. Saxe and K. Berlin, "Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features," *10th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 11–20, Oct. 2015.

[5] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware Detection by Eating a Whole EXE," *AAAI Workshops: Artificial Intelligence for Cyber Security*, 2018.

[6] D. Ugarte-Pedrero, I. Santos, B. Sanz, C. Laorden, and P. Bringas, "Countering Malware Analysis Evasion Techniques: A Case Study with Dynamic Analysis," *International Journal of Information Security*, vol. 18, no. 2, pp. 207–226, Apr. 2019.

[7] R. Xu, X. Wang, and J. Wu, "Classification Based Hard Disk Drive Failure Prediction: Methodologies, Performance Evaluation and Comparison," *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pp. 189–195, Aug. 2022, doi: 10.1109/CASE49997.2022.9920134.

[8] J. Kinder, "Towards Static Analysis of Obfuscated Binaries," *International Conference on Computer Aided Verification*, pp. 274–289, Springer, 2012.

[9] A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection," *23rd Annual Computer Security Applications Conference (ACSAC)*, pp. 421–430, Dec. 2007.

[10] J. Z. Kolter and M. A. Maloof, "Learning to Detect and Classify Malicious Executables in the Wild," *Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, Dec. 2006.