Grenze International Journal of Engineering and Technology, June Issue

AI-Powered Malware Deobfuscator

Shaurya Jain[1], Saiyam Verma[2], Salil Singh[3], Yash Paliwal[4] and Dilip Bharti[5]
[1-5]ABES Engineering College/ CSE - Data Science, Ghaziabad, India
Email: shaurya.22b1541119@abes.ac.in, saiyam.22b1541204@abes.ac.in, salil.22b1541162@abes.ac.in
yash.22b1541095@abes.ac.in, dilip.bharti01126@gmail.com

***Abstract*— This paper presents the AI-Powered Malware Deobfuscator which is a highly sophisticated and fully containerized security platform that is capable of detecting hybrid threats. The three key security processes are operating in coordination to make the system work, with the first being a static machine learning (ML) pipeline, which is a regression - based Scikit-learn random forest model that classifies Windows PE files based on extracted features using the pefile library; the first is distinguished as having helpful and understandable feature priority rankings; the third is a specific browser extension which blocks web threats dynamically by enforcing a real-time domain blocklist; and the second is non-ML, deterministic signature path, which critically uses the EICAR test string to test the operational integrity The architecture of the whole solution is highly reproducible, with each component coordinated easily by Docker Compose, and an application written in React so the user interface and scan status display are modern, and a back-end written with Fast API so that it can provide every necessary service in a structured, self-sufficient, and reliable way. Notably, zero-vector fallback mechanism to gracefully deal with non-PE and other artifacts out-of-domain ensures that the system is robust. The research provides a comprehensive pedagogical framework including the technical blueprint, engineering approaches, and threat model. It is supported by a rigorous examination procedure comprising of end-to-end functional assessment and offline measures. The conclusion of the work provides a future development roadmap where more advanced functions such as dynamic behavioral analysis and multi-modal content inspection are taken into consideration.**

***Index Terms*— Malware detection, Portable Executable (PE), Static analysis, Random Forest, Hybrid security, Fast API, React, Browser extension, URL blocking, EICAR, Explainability, Reproducibility, Software supply chain, Usability, Threat intelligence.**

I. INTRODUCTION

The development of malware detection has seen a high level in the last twenty years. The earliest antivirus had basically followed the signature-based detection techniques, where file hashes or known byte patterns were compared to a database of known malware signatures. Although signature-based detection is very reliable in the detection of known threats, it does not detect new or variant malware that employ obfuscation, polymorphism or packing measures to escape detection.

State of the art malware detection mechanisms are thus becoming more and more dependent on machine learning and on behavioral analysis and threat intelligence feeds. Without actually executing the code file, a statical analysis of machine learning can determine the type of files that can be executed by its structural characteristics, which is safer and quicker than dynamic analysis.

Grenze ID: 01.GIJET.12.2.69_20

Nevertheless, even the analysis of the code can be insufficient to deal with highly obfuscated malware. Dynamic analysis offers a better understanding of behavior but needs sandbox and increased calculations.
Hybrid security architectures are used to enhance reliability and coverage by using several methods of detection. Such systems normally combine signature detection, machine learning classification and policy implementation like domain blocking. These hybrid systems offer layered security in that the one layer is used to fill in the shortcomings of the other.

II. Background and Related Work

There are various methods that are used in malware detection research and they are signature -based, static, dynamic and machine learning based classification.

- Non-executable analysis Static malware analysis is the analysis of executable files without their execution. PE files have their headers and sections, which denote the layout of memory, entry points, imports, and other structural data.
  Machine learning models can be trained using these features to separate malicious and benign files. Malware detection research in machine learning has been made possible on large scale with public datasets including EMBER and SOREL-20M.
- Randem Forest and Gradient Boosted Trees are popular machine learning models based on trees, which are suited to malware classification since they are effective on structured numerical attributes, do not need much preprocessing, and offer feature importance to be interpretable.
- Detection Signature-based detection is still a significant part of any detection system of malware due to its provision of deterministic detection of known threats. It is well known that the EICAR test file is a harmless test artifact for antivirus applications and can test malicious detection workflows without actually transferring malicious code.
- Dynamic malware analysis is the process of running suspicious code inside sandbox, and observing the behavior (e.g., API calls, changing the registry, file system, and network traffic). Dynamic analysis is more informative but consumes more computing resources and may be avoided by malware identifying sandboxing.
- Another security mechanism is URL and domain blocking. Phishing websites or malicious downloads hosted on established malicious domains are often the beginnings of many attacks. Blocking of the known malicious domains help in limiting the exposure of users to threats and is used to complement malware detection systems based on files.
- Hybrid malware detectors are those systems that detect malware by combining the use of a static analysis, machine learning, signature detection, and policy enforcement in an effort to increase detection accuracy and system reliability.

III. Threat Model

The threat model provides the nature of attacks that the system is configured to identify and restrictions of the system.

A. In-Scope Threats

The system will identify malicious windows portable executable files and block access to the known malicious domains. Other elements of the system are signature detection by EICAR test file to verify detection workflows.

B. Out-of-Scope Threats

The system fails to inspect script based malware, e.g. the use of PowerShell or Java Script, Macro -enabled Office documents, fileless malware, highly packed or obfuscated binaries, encrypted network traffic, or attack malware using adversarial machine learning. These spheres are viewed as the future work.

C. Design Assumptions

This system has the following assumptions: uploaded files are small enough to be processed in memory, machine learning model can generalize through the training dataset distribution, domain blocklist is curated by hand. All files not of PE are considered out-of-scope and are processed safely by fallback mechanisms.
Having a clear scope of the system allows one to avoid overconfidence and the user will be aware of limitations of the system.

IV. System Architecture

The architecture of the system is modular which consists of frontend, backend, machine learning model, feature extraction module, browser extension and container orchestration.
The frontend interface is created in React, and it enables users to upload files, start scanning, and see the outcomes. The API is based on FastAPI and receives file uploads, feature extraction, model inference and URL analysis. PE header features are used to train a machine learning model which is a Random Forest classifier. The extract features processor relies on pefile library to interpret PE headers and extract numbers. The browser extension tracks navigation events, and blocks malicious domains based on a backend blocklist. Docker is the one that is employed in container orchestration and reproducibility across the environments.

A. File Analysis Workflow

In the case of uploading a file, the backend initially verifies the file to ensure that it does not contain the EICAR signature. Once the signature is found the file is considered as malicious and the machine learning model is not executed. In case of failure to identify the signature, the system tries to read the file as a PE file and identify header characteristics. In case of parsing error, the system has a zero-vector fallback feature set. The feature vector is subsequently modeled with the Random Forest model to classify and the outcome is sent back to the frontend.

B. Domain Policy Workflow

The browser extension tracks the navigation and gets the domain out of the URL. The domain is forwarded to the API server which determines whether the domain is on the malicious domain blocklist. In case the domain is malicious, the extension will block the page and redirect the user to the warning page.
This architecture guarantees modularity, extensibility and reproducibility.

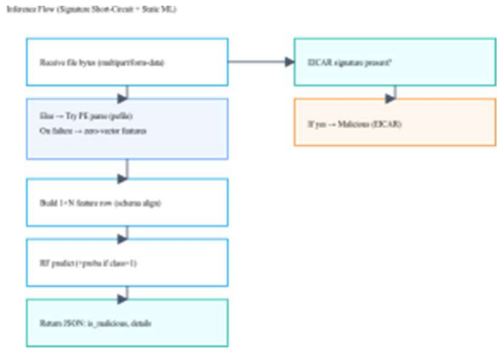

Fig 1. Interaction Diagram

V. Feature Engineering

The point of feature engineering is the extraction of structural information of Portable Executable files. The system derives characteristics of several PE header formats such as DOS header, file header, optional header, stack size, heap size, entry point address, image size, subsystem information and section statistics.
These structural characteristics are useful in detecting anomalies that are typically present in malware including abnormal entry points, section sizes, suspicious memory allocations sizes or unusual subsystem values.
When a file is not a valid PE file, the system creates a zero-vector feature set that has feature values in the form of zero. The Random Forest model is trained to understand that zero vectors are non -PE files which should be labeled as benign in training. This built-in backup system avoids crashes in the system and enhances resiliency when dealing with file types that are not supported.
The same feature schema that is used in training and inference guarantees that a mismatch in feature sequence will not lead to erroneous predictions.

VI. Model Development

This system operates on a machine learning model of a Random Forest classifier. The random forest models would be applicable to structured numerical data and they can give good classification results besides giving feature rankings that are important to interpret.

The training data will be made up of malicious and benign PE files. Training, validation, and test sets are used to test the performance of the models adequately. The number of trees, minimum samples per split, and the method of feature selection were some of the hyperparameters that were tuned with the help of the grid search to maximise the F1 score in the case of malicious classification.
The training process also involves the extraction of features in the labeled PE files, storing the schema of features, training the Random Forest model, testing the model, and saving the trained model to be used in inference.

TABLE I: DATASET COMPOSITION

Split	Benign	Malicious	Total
Train	1700	1500	3200
Validation	8000	8000	16000
Test	12000	12000	24000

Fig 2. Example confusion matrix on held-out test set

VII. INFERENCE PIPELINE

The inference pipeline takes uploaded files and generates the results of file classification. In the case of uploading a file, the system searches the EICAR signature. In the case of the signature existence, the file is defined as malicious immediately. Otherwise, the system tries to interpret the file as a PE file and takes out header characteristics. In case of parsing failure, a zero-vector feature set is created by the system.
The feature vector is transformed into a dataframe in proper order of features and it is fed into the Random Forest model. The model will be predictive of a malicious or benign file. In case the file is treated as malicious, the model confidence score is also provided by the system. The outcome is then sent back to the frontend interface.
The inference pipeline is also meant to be resilient, quick, and secure since files are never run.

VIII. URL POLICY ENFORCEMENT

The browser extension is a domain blocker that applies to the navigation events of the browser. On visiting any web site, the extension will fetch the domain and forward it to the back -end API to be analyzed. The backend does a query to the list of malicious domains. In case the domain is a malicious one, the extension blocks the page and redirects the user to a warning page. Blocking domain name rather than the full URL helps cut down on the maintenance cost and ensures that the attackers do not circumvent blocking by using alternative URLs. Some of the details that should be improved in the future are automated threat intelligence feeds, domain reputation scoring, and telemetry -based domain scoring.

XI. PERFORMANCE EVALUATION

Performance testing involves latency testing, throughput testing and resource usage testing. The system can classify PE files a dozen or more milliseconds according to latency measurements. Non -PE files are even faster to process since the zero-vector fallback does not need to perform feature extraction. The EICAR test file is the quickest detection route as it does not involve machine learning inference.
Throughput testing demonstrates that it is possible to sustain dozens of requests per second on a single worker. It is possible to horizontally scale the system with more than one container behind a load balancer.

The system uses the static analysis and runs no files, hence low resource usage. The machine learning model and backend service are the key reasons why the machine uses memory.

TABLE II: LATENCY AND THROUGHPUT

Scenario	Median (ms)	P95 (ms)
Benign PE	42	95
Malicious PE	45	100
Non-PE File	18	40
EICAR File	8	12

X. LIMITATIONS AND FUTURE WORK

There are a number of limitations to the system at present. It can only analyze Portable executable files, but not documents, scripts, or fileless malware. It is based solely on static analysis, and is unable to identify highly obfuscated malware. The domain blocklist is manually maintained and can go out of date. The system is also not much explainable and lacks model drift detection telemetry.
The addition of dynamic sandbox analysis, multi-modal malware detection of documents and scripts, explainable machine learning dashboards, automated ingestion of threat intelligence, model lifecycle
management, adversarial robustness testing and reputation-based domain scoring are all part of the future work.

XI. CONCLUSION

The AI-Powered Malware Deobfuscator offers an integrated malware detection framework that combines machine learning-based classification with signature-based detection as well as domain policy enforcement into a single platform. The system is focused on safety: every program line and file is analyzed in a static way, and each file is processed using memory only, reproducibility: the system deploys the program into a container, and usability: the program has a clear user interface. Random Forest model offers good protection of Portable Executable files and signature detection and domain blocking offer the extra -security measures. Future integration of dynamic analysis and multi-mode malware detection and automated threat intelligence is possible due to the modular architecture. This paper shows a viable and scalable hybrid malware detection platform with machine learning and conventional security measures.

REFERENCES

[1] Breiman, L. “Random Forests.” Machine Learning, 2001.
[2] Schultz, M. G., Eskin, E., Zadok, E., Stolfo, S. J. “Data Mining Methods for Detection of New Malicious Executables.” IEEE S&P, 2001
[3] Szor, P. The Art of Computer Virus Research and Defense. Addison-Wesley, 2005.
[4] Kolter, J. Z., Maloof, M. A. “Learning to Detect and Classify Malicious Executables in the Wild.” JMLR, 2006.
[5] Rieck, K., Trinius, P., Willems, C., Holz, T. “Learning and Classification of Malware Behavior.” DIMVA, 2008.
[6] Shafiq, M. Z., Tabish, M., Mirza, F., Farooq, M. “Peach: A Hybrid Feature Set for Malware Classification.” IEEE TDSC (early work 2009).
[7] Santos, I., et al. “Opcode Sequences for Malware Detection.” SECRYPT, 2010.
[8] Pedregosa, F., et al. “Scikit-learn: Machine Learning in Python.” JMLR, 2011.
[9] Egele, M., Scholte, T., Kirda, E., Kruegel, C. “A Survey on Automated Dynamic Malware Analysis Techniques and Tools.” ACM Computing Surveys, 2012.
[10] Microsoft. “Microsoft Malware Classification Challenge (BIG 2015).” Kaggle, 2015.
[11] Ribeiro, M. T., Singh, S., Guestrin, C. “Why Should I Trust You? Explaining the Predictions of Any Classifier.” KDD, 2016.
[12] Lundberg, S., Lee, S.-I. “A Unified Approach to Interpreting Model Predictions.” NIPS (SHAP), 2017.
[13] Saxe, J., Berlin, K. “eXpose: A Character-Level Convolutional Neural Network for Detecting Malicious URLs.” NDSS, 2017.
[14] Raff, E., et al. “Malware Detection by Eating a Whole EXE.” AAAI, 2018.
[15] Rosenberg, M., et al. “EMBER: An Open Dataset for Training Static PE Malware Models.” arXiv, 2018.
[16] Pendlebury, M., et al. “SOREL-20M: A Large Scale Benchmark Dataset for ML-based Malware Detection.” arXiv, 2020.
[17] NIST. “Secure Software Development Framework (SSDF).” SP 800-218, 2022.
[18] MITRE. “ATT&CK Framework.” (Accessed 2023).
[19] EICAR. “The EICAR Standard Anti-Virus Test File.” (Accessed 2024).