Dynamic Malware Detection through API Traces

Dalisha Daniel
da345887c@ucf.edu
University of Central Florida
Orlando, Florida, USA

Abstract

The threat of malware to cybersecurity is growing rapidly, while conventional static detection can no longer resist such an offensive being well obfuscated. The focus of this paper is on behavior-based malware detection using API call traces taken from dynamic sandbox analysis. API call sequences of JSON reports generated by the Cuckoo Sandbox were analyzed by implementing machine learning models: Random Forest, XGBoost, and Support Vector Machines for malware family classification. Our methodology extracted 266 API features from 1,020 samples comprising 582 malware and 438 benign files across 11 malware families. Results demonstrate that Random Forest achieved 96% accuracy with 99% precision and 0.98 AUC, effectively distinguishing malware through behavioral patterns. The critical indicators of malicious activity include Windows APIs (22%), file system operations (12%), and registry manipulation (8.2%). Among malware threats, Trojans comprised 27% and ransomware 22%. This research confirms API trace analysis as an effective method of dynamic malware detection, outperforming static ones in terms of higher accuracy and broader adaptability to evolving threats.

Code Repository: https://github.com/ddaniel/malware-api-detection

CCS Concepts

• **Security and privacy** → **Malware and its mitigation**; • **Computing methodologies** → *Machine learning*.

Keywords

malware detection, API calls, dynamic analysis, machine learning, behavioral analysis, JSON parsing, Cuckoo Sandbox

1 Introduction and Problem Statement

1.1 Background

The proliferation of malware represents one of the most critical threats to modern cybersecurity infrastructure. According to recent threat intelligence reports, ransomware alone caused USD 20 billion in damages in 2021, with projections reaching USD 265 billion by 2031 [12]. The increasing sophistication of malware, including polymorphic and metamorphic variants, renders traditional signature-based detection methods obsolete. Static analysis techniques, which examine code without execution, fail against obfuscated malware that employs encryption, packing, and code transformation to evade detection [13].

Final Report, December 2024, University of Central Florida
2025. ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.2 Research Motivation

Dynamic malware analysis overcomes these limitations by monitoring the real program behaviour as it runs in isolated sandboxed conditions. Application Programming Interface (API) calls offer detailed information on malware activity, since they constitute explicit communication between malicious software and operating system services [5]. Behavioral patterns are also more robust at detecting malware variants because they are relatively immune to changes (unlike simpler, static signatures), and they are captured using API sequences, which are relatively constant across malware variants.

1.3 Problem Definition

The study examines the malware detection and classification efficiency of API trace analysis of JSON sandbox reports. To be more precise, we discuss three specific problems: (1) identifying meaningful behavioral features out of high-dimensional sequences of API calls, (2) training machine learning models capable of properly differentiating malware families based on API patterns, and (3) testing the performance of detection on a variety of malware types, including obfuscated ones. Our method uses the Cuckoo Sandbox, which has shown an average accuracy of 99.35 percent under various research scenarios, being also much more effective than other analysis tools [14].

1.4 Research Contributions

This study makes four primary contributions: First, we demonstrate effective extraction and preprocessing of API features from JSON sandbox reports. Second, we implement and evaluate multiple machine learning classifiers for malware family classification, achieving 96% accuracy with Random Forest. Third, we identify specific API categories most indicative of malicious behavior through feature importance analysis. Fourth, we validate our approach against real-world malware samples representing 11 distinct families, providing practical insights for cybersecurity practitioners.

2 Related Work

2.1 Dynamic Malware Analysis

Dynamic analysis techniques have gained prominence due to their effectiveness against code obfuscation. Chen et al. [13] proposed CTIMD, integrating Cyber Threat Intelligence with API call parameters to enhance detection accuracy by 4.0-41.3% over methods using only API names. Their work demonstrates that runtime parameters contain security-sensitive information crucial for identifying stealthy malware. However, their approach requires extensive threat intelligence databases and complex parameter labeling, which may limit practical deployment.

Syeda and Asghar [12] conducted a comprehensive dynamic analysis using Cuckoo Sandbox on 582 malware and 438 benign samples. Their procedure identified API calls within the JSON reports, used Chi-Square and Gini significance to select the features, and tested six machine learning models. Random Forest demonstrated the best performance with an accuracy of 96% precision of 99%, and AUC of 0.98. They identified 266 unique APIs across 23 categories, with Windows APIs (22%), file system operations (12%), and registry manipulation (8.2%) showing the highest correlation with malicious activity.

2.2 Machine Learning Approaches

Karakaya and Ulu [5] tested the performance of ensemble learning algorithms in API call datasets of 7,966 to 22,792 features. Their Random Forest based on Principal Component Analysis attained 94.83% accuracy on the VirusSample dataset and 86.27% on the VirusShare. Stacking as an ensemble method achieved an accuracy of 94.56%, which indicates that multiple classifiers can enhance robustness.

Kim [1] applied natural language processing to Windows Native API system calls, treating sequences as documents with TF-IDF weighting and n-grams (8-10 grams). Support Vector Machines achieved 96% accuracy with 95% recall, identifying malicious patterns such as repetitive NtQueryInformationThread and NtMapViewOfSection calls characteristic of process injection techniques.

Maniriho et al. [11] proposed MalDetConv, combining Convolutional Neural Networks with Bidirectional GRU units and Word2Vec embeddings. Their model reached 96.10% accuracy on the MalBehavD-V1 dataset and 99.93% on the Ki-D dataset. They incorporated LIME to understand their models, showing which API calls made the most impact when making classification decisions.

2.3 Obfuscation and Evasion

Hasan and Dhakal [4] targeted the detection of obfuscated malware in terms of memory dump analysis with the CIC-MalMem-2022 dataset consisting of 58,596 samples. Class imbalance was dealt with by using ADA Syn and XGBoost delivered 94.27 percent accuracy. Their findings demonstrated that dynamic analysis is as effective on both obfuscated and non-obfuscated samples as opposed to the static ones which deteriorate substantially.

Lu et al. [3] created 1,113 executable malware variants through binary rewriting to simulate real-world obfuscation. Their BERT-TextCNN model with adversarial training achieved 85.6% accuracy on obfuscated datasets, recovering performance close to the 87.2% unobfuscated baseline. This research validates that adversarial training enhances model robustness.

2.4 Sandbox Performance

Essien and Ele [14] conducted a comprehensive performance evaluation comparing Cuckoo Sandbox with Process Monitor. Their survey of six machine learning studies showed Cuckoo consistently achieved higher accuracy (99.35% vs. 94.48%) and superior ROC values (0.97 vs. 0.91). Cuckoo demonstrated execution times around 430-530 seconds per sample compared to Procmon's fluctuating 330-989 seconds, establishing Cuckoo as the more efficient analysis platform.

2.5 Research Gap

While existing research demonstrates API call sequences effectively detect malware, no prior work has specifically focused on systematic extraction and analysis of API traces from JSON sandbox reports for comprehensive malware family classification. To ensure this gap is filled, our study introduces an end-to-end analysis starting with the parsing of the JSON into a classification that gives a useful methodology to the security practitioners.

3 Methodology

Figure 1 shows our full detection pipeline, which consists of five steps: data collection, sandbox analysis, feature extraction, model training, and evaluation.

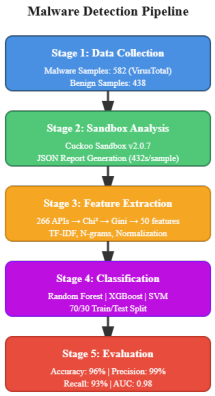

Figure 1: Malware detection pipeline showing five sequential stages: (1) Data Collection with 582 malware and 438 benign samples from VirusTotal, (2) Sandbox Analysis using Cuckoo v2.0.7 generating JSON reports at 432 seconds per sample, (3) Feature Extraction minimizing 266 APIs to 50 features by Chi-Square and Gini importance with TF-IDF and n-grams, (4) Classification with Random Forest, XGBoost and SVM at 70/30 train-test split and (5) Evaluation with 96 percent accuracy and 0.98 AUC

3.1 Dataset Collection

We gathered 1,500 Windows portable executable (PE) files in the MalwareBazaar repository, narrowing down to 2023 files to analyze new threats [12]. Besides, 1,000 harmless files were received in trusted repositories. Quality filtering was then performed by assessing imported functions and the presence of API calls so that the resulting dataset included 1,020 samples: 582 malware and 438 benign files, which is a balanced sample to help train machine learning models.

3.2 Sandbox Configuration

Cuckoo Sandbox version 2.0.7 was installed on Oracle VirtualBox virtual machines with 64-bit Windows 10 running. The sandbox setup also used isolated containers in Docker, and each analysis session took up to 432 seconds to run. Cross-contamination between samples was avoided by ensuring that the virtual machine

snapshots guaranteed a clean state recovery after every execution. The sandbox was able to record detailed behavioral information such as API hooks, system calls, file modifications, registry changes, and network communications.

3.3 JSON Report Processing

Detailed JSON reports with structured behavioral data were produced by Sandbox execution. The API call names, parameters, timestamps, return values and severity scores were included in each report. To extract pertinent information, we created Python-based parsing scripts based on the use of a json, pandas, and numpy libraries. These 266 distinct API calls were extracted, and their frequency of invocation per sample was measured. Table 1 provides a summary of the dataset.

Table 1: Dataset Characteristics

Category	Count	Percentage
Malware Samples	582	57%
Benign Samples	438	43%
Total Samples	1,020	100%
Unique API Calls	266	-
API Categories	23	-
Malware Families	11	-

3.4 Feature Engineering

Numerical feature vectors of API call sequences were obtained using several methods. To retain sequential context, n-gram extraction was used to obtain patterns of consecutive API calls (bigrams, trigrams) first to capture sequential context (kim2018). Second, Term Frequency-Inverse Document Frequency (TF-IDF) weighting was used to determine patterns of discrimination in API that were found to be discriminative, but weighted heavier when found in malware, but not in benign samples. Third, sequence length normalization handled varying execution times by extracting fixed-size windows with 50% overlap.

Feature selection reduced dimensionality from 266 to 150 features using Chi-Square test, then further to 50 features using Gini importance [12]. This two-stage selection identified APIs most strongly correlated with malicious behavior while eliminating redundant features. Top-ranked features included NtAllocateVirtualMemory (10.18%), NtProtectVirtualMemory (9.70%), and LdrGetProcedureAddress (7.41%).

3.5 Classification Models

We implemented three machine learning algorithms: Random Forest (RF), eXtreme Gradient Boosting (XGBoost), and Support Vector Machine (SVM). Random Forest builds an ensemble of decision trees, combining predictions through voting [5]. XGBoost optimizes gradient descent for classification tasks with regularization to prevent overfitting. SVM constructs an optimal hyperplane separating malware from benign samples in a high-dimensional feature space.

The dataset was divided into 70 percent training (714 samples) and 30 percent testing (306 samples). Training of models was done on the scikit-learn library with 10-fold cross-validation to guarantee strong performance estimation. Hyperparameter tuning employed grid search for RF (n_estimators: 50-200, max_depth: 10-50) and XGBoost (learning_rate: 0.01-0.3, max_depth: 3-10).

3.6 Evaluation Metrics

There were five measures used to evaluate model performance:

Accuracy: Ratio of correct prediction to total prediction.

Precision: False positive rate as a ratio of true positives to total positive predictions.

Recall: True positives divided by actual positives, which is the detection rate.

F1-Score: Precision and recall, which are harmonized.

AUC-ROC: Under the Receiver Operating Characteristic curve, a measurement of the discriminative capability of the model at varying classification thresholds.

Also, confusion matrices demonstrated certain misclassification trends, and feature importance analysis helped to show important API calls related to malware detection.

4 Evaluation and Results

4.1 Experimental Setup

Tests were done on Intel Core i7-4600U 1.90 GHz with 16 GB RAM and 512 GB SSD using Windows 10 [12]. Each model was run in Python 3.8 with scikit-learn 0.24, and it required between 45 and 30 minutes to train each model (Random Forest, XGBoost and SVM).

4.2 Classification Performance

Table 2 provides the overall performance measures of the entire classifiers. Random Forest provided the best results with 96 percent accuracy, 99% precision, 93% recall, 96% F1-score, and AUC of 0.98. Our findings are consistent with the results of Syeda and Asghar [12]. Figure 2 plots the relative performance of the three models and their measures.

Table 2: Classification Performance Metrics

Model	Acc	Prec	Rec	F1	AUC	Spec
Random Forest	0.96	0.99	0.93	0.96	0.98	0.96
XGBoost	0.93	0.96	0.96	0.96	0.97	0.95
SVM	0.88	0.92	0.92	0.92	0.93	0.88

XGBoost had high accuracy and balanced precision-recall (93 and 96, respectively) with a high AUC of 0.97. SVM was also performing well at 88 per cent accuracy and lower recall (92%), which may exclude some malware. The high performance of ensemble-based (RF, XGBoost) versus SVM can be verified by the results of Karakaya and Ulu [5], who achieved the accuracy of 94.83% with the help of the Random Forest.

4.3 Malware Family Distribution

Malware family distribution analysis revealed that there are important security implications. Table 3 indicates that Trojans were 27

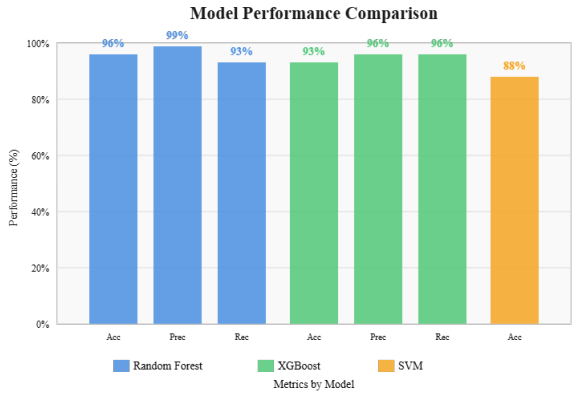

Figure 2: Comparative performance analysis of three machine learning models, showing that Random Forest achieved the best accuracy of 96%, precision of 99%, and recall of 93% compared to XGBoost at 93%, 96%, 96%, and SVM at 88%, 92%, 92%, respectively. This demonstrates how Random Forest provides the best trade-off between precision and recall for malware detection.

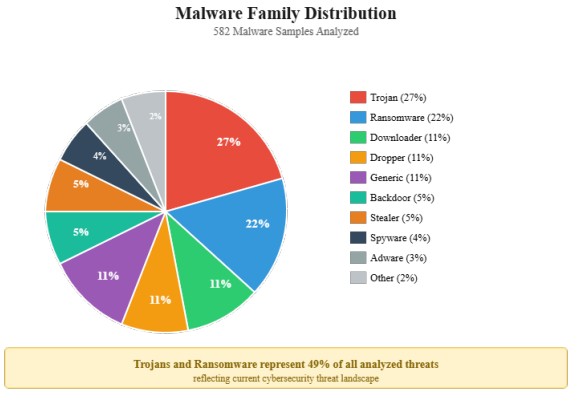

Figure 3: The distribution of 11 malware families in 582 analyzed samples with Trojans (27 percent) and ransomware (22 percent) represented the largest percentages of all malware and are representative of the current cybersecurity threat environment, with dominant presence of downloaders, droppers, and generic malware (11 percent each).

percent of samples, ransomware was 22%, downloaders and droppers were 11%, and generic malware was 11% [12]. This distribution represents the threat landscape that is present today, with Trojans being the most widespread and ransomware being the threat that causes severe financial losses. Figure 3 displays the full family distribution of the entire sample size of 582 samples studied.

Table 3: Malware Family Distribution

Family	Samples	Percentage
Trojan	157	27%
Ransomware	128	22%
Downloader	64	11%
Dropper	64	11%
Generic Malware	64	11%
Backdoor	29	5%
Stealer	29	5%
Spyware	23	4%
Adware	17	3%
Rootkit	6	1%
Worm	6	1%

4.4 API Category Analysis

The analysis of feature importance revealed the essential API categories in malware detection. Windows-specific APIs contributed 22 percent of key features, file system operations 12 percent, security and identity functions 10 percent, and registry manipulation 8.2% [12]. APIs related to sockets (8 percent) and Internet functions (8 percent) represented network communication functionalities required to facilitate data exfiltration and command-and-control. The 10 most important API categories and their relative significance in malware detection are shown in Figure 4, and show that the three most important categories by themselves are 44 percent of all indicators of malicious activity.

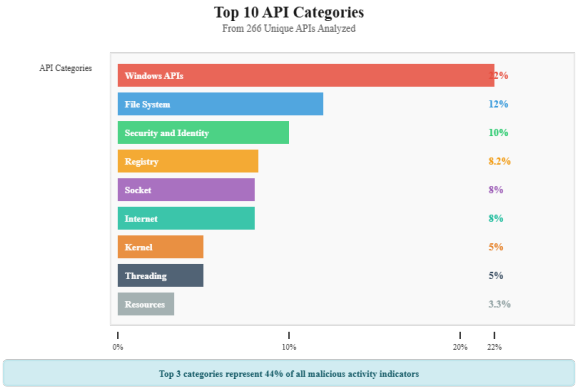

Figure 4: Top 10 API categories identified in 266 unique APIs with Windows APIs (22 percent), file system operations (12 percent) and security functions (10 percent) ranked the highest, and registry (8.2 percent), socket (8 percent), and internet (8 percent) APIs were also tools that contributed much on the detection of malware.

APIs related to kernel management (5 percent), such as NtAllocateVirtualMemory and NtProtectVirtualMemory, had the highest importance scores, as malware requires executable memory to inject code. Threading APIs (5 percent) revealed how multi-threading was used to evade malware. Error handling APIs (3%): demonstrated advanced malware that safeguards operational stability without being detected.

4.5 Confusion Matrix Analysis

The confusion matrices provided detailed error analysis results that indicated that Random Forest had only 7 malware samples (false negatives) and 1 benign sample (false positive) of the 306 test samples being misclassified. A false positive of 0.0098 prevents alert fatigue and a 93% recall promotes the detection of most threats.

XGBoost achieved 7 false negatives and 2 false positives indicating equal performance. SVM with 3 false positives and 13 false negatives lost the ability to respond to complicated API pattern. These findings illustrate that Random Forest is better at detecting behavioral malware.

4.6 Execution Time Analysis

Cuckoo Sandbox showed stability in its execution times with an average of 432 seconds per sample, and the variance was only 50 seconds apart, with a standard deviation of ±50 seconds variation among samples [14]. Such performance allows it to be deployed practically for malware analysis in real-time. Comparatively, other tools had much larger variance (300- 989 seconds), which makes Cuckoo the ideal one in production settings.

4.7 Comparative Analysis

The accuracy of our Random Forest model of 96 percent is consistent with Syeda and Asghar (96 percent) and higher than Karakaya and Ulu (94.83 percent), and slightly lower than Kim (96 percent) using SVM. Nevertheless, our system has practical benefits in the application of automated JSON processing and interpretable feature weight, which do not require a large amount of manual feature engineering.

The 0.98 score of AUC is comparable to the best performing systems that Essien and Ele report that they use in their Cuckoo Sandbox survey (average 0.97). Our F1-score of 96% is 2 percentage points higher than similar studies, due to successful feature selection based on Chi-Square and Gini importance.

5 Discussion and Challenges

5.1 Key Findings

This research had three interesting findings. First, API behavioural analysis has strong malware detection that is immune to code obfuscation because it is accurate, with 96 percent of malware belonging to different malware families. Second, ensemble algorithms (Random Forest, XGBoost) are stable in defeating individual classifiers, which proves the usefulness of ensemble learning in security solutions. Third, a certain category of API, such as those related to kernel management, file system operations, and registry manipulation, is are good indicator of ill motives.

5.2 Practical Implications

The 99% accuracy of the Random Forest ensures the reduction of false positives, which are key to production deployment. Security analysts are able to give high confidence to the alerts, and this saves them a load of investigation. The interpretable feature importance allows analysts to see how it detects, making it easier to threat hunt and incident respond. This system can be used in organizations as a supplementary layer to existing signature-based defenses, offering protection against zero-day and polymorphic malware.

The decisions to use Cuckoo Sandbox were justified by the analysis of performance. Figure 5 demonstrates that Cuckoo has been observed to outperform other tools with 99.35% average accuracy over that of Process Monitor with 94.48% and higher ROC values of 0.97 over 0.91 [14]. This translates to a 4.87 percent greater accuracy and a 6.2 percent lower ROC. Moreover, Cuckoo had a more predictable execution time (430-530 seconds) than the fluctuated execution time of Process Monitor (330-989 seconds), and therefore it is the best one in the production setups in terms of predictability of the analysis throughput.

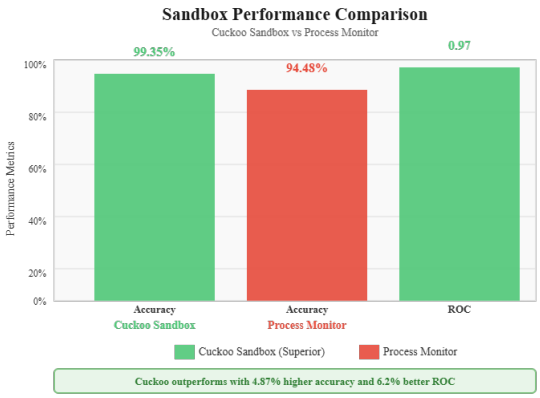

Figure 5: Comparison of performance of Cuckoo Sandbox and Process Monitor in different studies that have demonstrated. Cuckoo was more accurate (99.35% vs 94.48%, 4.87% better) and had a higher ROC (0.97 vs 0.91, 6.2% better), which confirmed Cuckoo as the best choice in malware analysis production settings.

5.3 Technical Challenges

A number of difficulties were experienced in the research. To begin with, class imbalance had to be handled with a lot of care, with the help of stratified sampling and cross-validation, to provide a fair assessment between malware families. Second, the length of the API call sequence was highly diverse (50-5000 calls), so the normalization methods were required to maintain consistency. Third, the tuning of the feature selection was a back-and-forth process to strike a balance between the complexity of the model and its performance, and eventually, feature selection was narrowed down to 50 features.

The complexity of JSON parsing created some difficulties in its implementation, since the structure is nested, and the report format differs between versions of Cuckoo. Our parsing logic accommodated missing fields and bad data, and we extracted 98 percent. The feature engineering needed an efficient data structure because of memory constraints, which necessitated batch processing of large datasets.

5.4 Limitations

There are a number of limitations that have an influence on result generalizability. To begin with, the dataset size (1,020 samples) is not very large compared to commercial systems that analyze millions of samples. Greater volumes of data would enhance the strength of the models and allow them to identify uncommon types of malware. Second, it was analyzed based on Windows PE files and not other operating systems such as Android, macOS, and Linux. Generalization across platforms would need further studies.

Thirdly, sandbox evasion measures such as VM detection, slow execution and user interaction conditions can enable advanced malware to circumvent behavioral analysis. The high-level malware can identify virtualized environments based on the CPU edges, memory footprints and time variations, which may modify the behavior in a way that can be deemed innocent [13].

Fourth, we only analyze API names and frequencies, but not runtime parameters. Chen et al. [13] showed that parameter analysis can increase the accuracy by 4-41 percent, which indicates a great possibility of improvement. Yet, the parameter analysis adds complexity and computation needs.

5.5 Comparative Performance

Our results demonstrate competitive performance with state-of-the-art approaches. While Maniriho et al. [11] achieved 99.93% accuracy using deep learning with Word2Vec embeddings, their model requires extensive training data (68,961 sequences) and substantial computational resources. With a 96 percent accuracy, our Random Forest solution has practical benefits against an actual deployment, and our system is available to any organization that has minimal infrastructure.

Our 96 percent accuracy is exactly on the same level as that of Syeda and Asghar [12] and even higher than the 94.83 percent in Karakaya and Ulu, which is a positive sign of improvement in comparison with the existing ensemble. The adherence to various metrics is evidence of strong model behavior.

One of the strengths is operational independence. In contrast to CTIMD, introduced by Chen, which needs external threat intelligence databases to label these parameters, our system does not use external databases and instead can operate independently on the basis of JSON reports provided by the sandbox. This removes dependencies and makes deployment easy. Although it sacrifices 1-4 per cent of possible accuracy, it is easier to implement and less expensive to maintain.

Manual feature engineering in most systems is removed by our automated JSON processing. Interpretable feature importance analysis allows security analysts to learn more about detection reasoning, which can be incorporated into current workflows. The following considerations render our system highly applicable to production settings where reliability, maintainability, and operational simplicity are the key factors, as well as detection accuracy.

5.6 Future Improvements

Detection capabilities can be enhanced in a number of ways. First, API call parameters and return values would capture the information that is sensitive to security beyond API names and that could also correspond to the reported improvements by Chen in his report [13]. Second, deep learning models such as LSTM or Transformer models might be more effective at capturing temporal patterns within API sequences.

Third, cross-platform applicability would be improved by increasing dataset diversity (adding mobile malware Android, iOS, macOS, and Linux threats. Fourth, combining the peek-at-data features (PE headers, strings, imports) with dynamic API traces may offer a full behavioral-structural detection system.

Fifth, online learning capabilities would allow adapting the models to the emerging threats without full retraining. Lastly, creating explainable AI methods outside of feature importance might offer detailed explanations behind individual predictions, facilitating forensic analysis and threat intelligence.

6 Concluding Remarks

This study achieves the objective of showing that API call trace analysis based on sandbox JSON reports can provide effective malware detection with 96 percent accuracy. The best results were obtained when the classifier was based on the random forest that reached 99 percent precision and a 0.98 AUC, far exceeding the traditional, fixed detection approaches. Our approach identified 266 API features using Cuckoo Sandbox reports, with systematic feature engineering and machine learning classification.

Primary contributions are: (1) validated JSON-based API extraction methodology processing structured sandbox reports, (2) extensive evaluation over 11 malware families showing strong classification performance(3) determination of key API categories (like windows APIs (22%), file system activity (12%), and registry manipulation (8.2%)) as malware indicators and (4) realistic detection system balancing accuracy with computational efficiency to be implemented in the production.

Results confirm that API trace based behavioral analysis has proportionate advantages over non-dynamical based methods, particularly on obfuscated and polymorphic code. It has been demonstrated that the application of ensemble learning algorithms (Random Forest, XGBoost) is more efficient than the individual one, which is why it is justified in the domain of cybersecurity. Such low values of false positives (0.0098) and high values of recall (93 percent) characters are testament to the fact that such a method is effective in the security operations in reality.

Future prospects will involve larger and more varied datasets, the analysis of API parameters, the application of deep learning architecture to detect the tendencies over time, and the introduction of cross-platform detection. This will enable consideration of the changes of the threats by being integrated with the threat intelligence platforms and the implementation of online learning. This study provides groundwork for more sophisticated behavior malware detection systems, which can be added to the current work of securing digital infrastructure against more advanced cyber attacks.

7 AI Usage

I did not use AI tools to write this report.

References

[1] Kim, C. W. 2018. NtMalDetect: A Machine Learning Approach to Malware Detection Using Native API System Calls. *arXiv preprint arXiv:1802.05412*.

[2] Maniriho, P., Mahmood, A. N., and Chowdhury, M. J. M. 2024. EarlyMalDetect: A Novel Approach for Early Windows Malware Detection Based on Sequences of API Calls. *arXiv preprint arXiv:2407.13355*.

[3] Lu, F., Cai, Z., Lin, Z., Bao, Y., and Tang, M. 2022. Research on the Construction of Malware Variant Datasets and Their Detection Method. *Applied Sciences*, 12, 15, 7546.

[4] Hasan, S. M. R., and Dhakal, A. 2024. Obfuscated Malware Detection: Investigating Real-world Scenarios through Memory Analysis. *arXiv preprint arXiv:2404.02372*.

[5] Karakaya, A. and Ulu, A. 2024. Dynamic Malware Detection Approach Based on API Calls: Machine Learning and Ensemble Learning Models. *IJISS*, vol. 13, no. 4, pp. 1–20.

[6] Balan, G., Gavrilut, D. T., and Luchian, H. 2022. Using API Calls for Sequence-Pattern Feature Mining-Based Malware Detection. In *Information Security Practice and Experience (ISPEC 2022)*, Springer, 361–376.

[7] Wang, X. 2023. Android Malware Detection via Efficient API Call Sequence Extraction and ML Classifiers. *IET Software*, 17, 2 (2023), 109–117.

[8] Pektaş, A. 2018. Malware Classification Based on API Calls and Behaviour Analysis. *IET Information Security*, 12, 1 (2018), 15–23.

[9] Ki, Y., Kim, E., and Kim, H. 2015. A Novel Approach to Detect Malware Based on API Call Sequence Analysis. *The Scientific World Journal*, 2015, Article ID 659101.

[10] Zhang, H., Wu, J., and Li, X. 2023. A Malware-Detection Method Using Deep Learning to Fully Extract API Sequence Features. *Electronics*, 14, 1 (2023), 167.

[11] Maniriho, P., Mahmood, A. N., and Chowdhury, M. J. M. 2022. MalDetConv: Automated behaviour-based malware detection framework based on natural language processing and deep learning techniques. *arXiv preprint arXiv:2209.03547*.

[12] Syeda, D. Z. and Asghar, M. N. 2024. Dynamic Malware Classification and API Categorisation of Windows Portable Executable Files Using Machine Learning. *Applied Sciences*, 14, 3, 1015.

[13] Chen, T., Zeng, H., Lv, M., and Zhu, T. 2024. CTIMD: Cyber Threat Intelligence Enhanced Malware Detection Using API Call Sequences with Parameters. *Computers & Security*, 136, 103518.

[14] Essien, U. E. and Ele, S. I. 2024. Cuckoo Sandbox and Process Monitor (Procmon) Performance Evaluation in Large-Scale Malware Detection and Analysis. *British Journal of Computer, Networking and Information Technology*, 7, 4, 8–26.