International Journal on Optimization and Applications
IJOA. Vol. 06, Issue No. 01, Year 2026, www.ijoa-journal.com

Development of an IDS/IPS based on machine learning technique to detect suspicious behavior in a network

Salma Doukkar
ENSA - Ibn tofail University
Salma.Doukkar@uit.ac.ma

Sophia ALAMI KAMOURI
ENSA - Ibn tofail University
s.alami@um5r.ac.ma

***Abstract*—As the cybersecurity landscape evolves, the development of advanced and efficient malware detection systems has become increasingly crucial. This research addresses the persistent challenge of identifying malware and safeguarding digital assets against cyber threats. Traditional detection methods often struggle to adapt to the dynamic nature of malicious activities [1], highlighting the need for innovative approaches that enhance detection accuracy while considering current cybersecurity challenges.**

This study focuses specifically on detecting malware through network traffic analysis. We propose a machine-learning-based framework for malware detection, emphasizing the effectiveness of two prominent algorithms: Random Forest (RF) and Support Vector Machine (SVM).

Our comparative performance analysis reveals that the model utilizing the Random Forest algorithm exhibits superior results. Notably, both classifiers achieve a True Positive Rate (TPR) exceeding 97%, while maintaining a False Positive Rate (FPR) of less than 4%. This indicates a robust capability in accurately identifying malware without generating excessive false alarms.

The model developed in this research has significant potential to assist organizations and cybersecurity professionals in anticipating and mitigating malware threats. By integrating this model into daily network operations, it can facilitate proactive decision-making and enhance the overall security posture of organizations. Additionally, our findings contribute to the broader field of cybersecurity by offering insights into effective machine-learning strategies for malware detection.

***Index Terms*—Machine learning, Malware Detection, Intrusion Detection, Malware Analysis, Random Forest, SVM**

I. Introduction

In today's digital age, many electronic devices face significant risks from malware. The term "malware" refers to malicious software specifically designed to damage or disrupt a target system. Malware can infiltrate networks, infect computers and other smart devices, steal sensitive data, damage vital infrastructure, and more [2].

These programmes include malware such as ransomware, rootkits, worms, spyware, bots, and viruses. According to [3], IT services claims that in only one year, one billion emails were exposed, impacting one in five internet users, and resulting in data breaches that cost organisations, on average, $4.35 million in 2022. The first half of 2022, there were about 236.1 million ransomware assaults worldwide.

In 2021, the accounts of one in two internet users in America were compromised and we can also give the example of the National Social Security Fund (CNSS), on April 8, 2025 in Morocco, which was the victim of a major cyberattack that resulted in the leak of sensitive data. This attack targeted the CNSS's information systems, resulting in a data leak concerning millions of citizens.

Malware attacks are becoming more complicated over time, despite improvements in detection, proper family class classification, and continual evolution, malware continues to be a serious threat to the internet [4]. Malware has also increased the risk of sophisticated attacks, such as multi-stage attacks [5]–[8] and Distributed Denial of Service (DDoS) attacks [9], [10], which have been a serious threat in recent years.

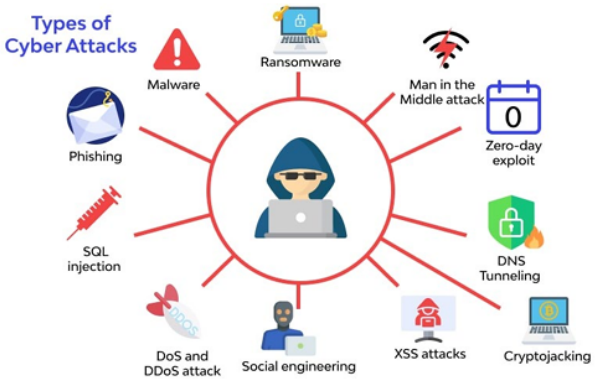

Fig. 1. Types of Cyber Attacks

To reduce the risk of cyberattacks, Intrusion Detection Systems (IDSs) are employed to monitor network traffic. The capability to detect viruses on a computer enables the development of malware prevention solutions, often incorporating unique signatures to identify infections. Malware can manifest in various forms, such as ransomware designed to extort money and spyware intended for surveillance.

Despite the advancement of several Machine Learning (ML) techniques for detecting anomalies in network traffic, human expertise remains crucial in creating many tools and methods. Specifically, handcrafted features play a vital role in traditional ML-based malware analysis methods. These features reflect what cybersecurity professionals consider the most important characteristics of malware. However, the feature engineering process can be labor-intensive, and the features created are often specific to particular tasks and subject to individual judgment.

Traditional ML methods have proven effective for malware detection, but they heavily rely on the knowledge and experience of security professionals to define the features characterizing malware. Research has shown a need for representations of malware that are less dependent on human expertise, addressing a significant yet unresolved challenge in the field.

This paper proposes a machine-learning-based approach for malware detection, with particular attention to the Random Forest (RF), Support Vector Machine (SVM).

The remainder of this paper is organised as follows. Section 2 presents the related to malware detection, section 3 describes the proposed methodology, section 4 shows the evaluation results, Section 5 presents a performance analysis of the utilised ML algorithms and Section 6 concludes the paper.

II. Related Work

With the rapid evolution of technology, malware threats and cybersecurity challenges are becoming increasingly difficult to counter. Malware, commonly referred to as a computer virus, is malicious software designed to infiltrate systems, damage files, and compromise user data. Some forms of malware, such as Trojans, disguise themselves as legitimate applications in order to gain unauthorized access and potentially take full control of a victim's computer [11].

The continuous emergence of malware variants and vulnerabilities in cloud environments raises the critical question of how assets can be effectively protected. To address this, various malware recognition techniques, particularly those leveraging machine learning, have been explored as promising solutions [12].

Machine learning offers significant advantages for classification tasks such as malware detection. By learning behavioral patterns and malicious activities, machine learning models can help mitigate the damage caused by malware. However, as highlighted in [13], detecting sophisticated and unpredictable malware remains challenging. Therefore, models must be trained and tested on diverse datasets containing a wide range of malware in order to improve detection accuracy, especially in high-risk scenarios involving destructive attacks. This demonstrates the need for rigorous and comprehensive evaluation.

Beyond malware detection, machine learning also provides useful applications in other domains, with several studies showcasing its versatility.

A. *Different Categories of Malware*

Different categories of malware are commonly discussed in the literature. For example, viruses are pieces of code that replicate until they corrupt files or system structures [14]. Worms spread automatically across networks to infect multiple machines. Trojans, disguised as harmless programs, trick users into downloading them before revealing their malicious intent.

Runtime malware may steal or encrypt data, often demanding ransom for recovery. Such attacks can be especially harmful to organizations managing sensitive customer information, as they can lead to data breaches, financial loss, and reputational damage.

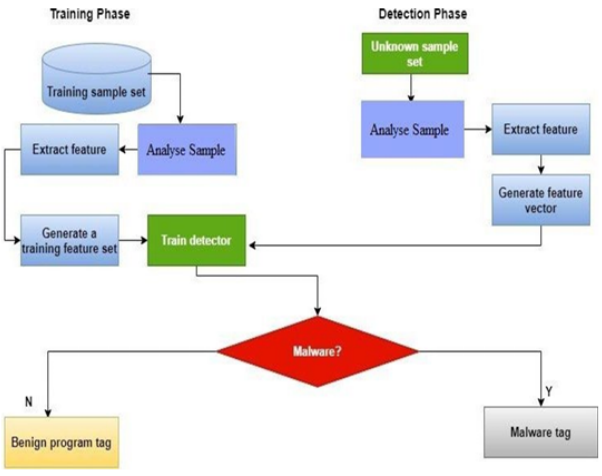

Fig. 2. Proposed ML malware detection method

Because malware is constantly evolving, rule-based and signaturebased detection methods are often insufficient. Signature-based detection identifies threats using

byte patterns stored in a database. While accurate for known threats, this approach fails against novel or polymorphic malware and requires frequent human updates.

Signature-less methods attempt to classify programs based on predefined suspicious behaviors (e.g., unusual assembly calls, unauthorized file permission changes, or abnormal file modifications). A threshold is then applied to determine whether the activity should be flagged as malicious. However, these approaches also require careful tuning and large datasets for effective training.

Anomaly detection techniques, capable of identifying new or unknown threats, offer another line of defense but suffer from high false-positive rates. Malware analysis is generally divided into two categories: static analysis, which inspects code without execution (allowing style recognition and code flow profiling), and dynamic analysis, which observes behavior during execution. Both approaches are widely used in research and industry.

Recent studies confirm that malware continues to grow in scale and sophistication, exploiting vulnerabilities in modern operating systems and even in browsers with built-in VPN services. These weaknesses are often leveraged by attackers, sometimes targeting inexperienced users, to launch severe cyberattacks that compromise system and network security.

III. Proposed Methodology

The proposed approach consists of several key stages:

A. Stage 1 – Environment Setup

The first step involves setting up the development environment.

B. Stage 2 – Log Generation with Anomalies

Synthetic log data containing both normal and anomalous events is generated. A custom Python script (`logs_generator.py`) is used to create a structured dataset, stored in CSV format, which serves as the foundation for the training and evaluation phases.

C. Stage 3 – Data Preprocessing

The generated logs are preprocessed to ensure quality and usability. This includes cleaning, normalization, feature extraction, and transformation into numerical vectors suitable for machine learning models.

D. Stage 4 – Model Training

Two supervised machine learning algorithms are employed: Random Forest (RF) and Support Vector Machine (SVM). Random Forest, an ensemble-based method, is chosen for its robustness and ability to handle high-dimensional data, while SVM is selected for its effectiveness in classifying anomalies in complex feature spaces. Both models are trained to distinguish between normal and abnormal log patterns.

E. Stage 5 – API Development

To enable real-time usability, an API is created to serve the trained models. This API allows external systems to send logs and receive classification results (normal or anomalous).

F. Stage 6 – Log Simulation

A simulation process is implemented to mimic the continuous flow of logs. The logs are sent through the API to test the model's ability to process and classify data streams in near real time.

G. Stage 7 – Deployment and Monitoring

Finally, the solution is deployed in a test environment with monitoring mechanisms in place to track model performance and adapt to changes in data patterns.

H. Evaluation Metrics

The performance of the models is assessed using standard metrics, including accuracy, precision, recall, and F1-score, to measure the detection quality of both algorithms.

IV. System Design and Workflow

A. Environment Setup

The development environment was prepared using Python and its scientific ecosystem. Key libraries such as NumPy and Pandas were used for data manipulation and numerical analysis, while Redis was integrated as an in-memory database for caching and session management. Celery handled asynchronous and background tasks, and Uvicorn served as the lightweight server for deploying Python applications. Additionally, other essential libraries, such as Matplotlib for data visualization and Scikit-learn for machine learning, were also utilized. This setup ensured scalability, efficiency, and smooth execution throughout the project.

B. Logs Generation

To create a realistic dataset for training and testing the anomaly detection models, a Python script was developed to generate synthetic logs containing both normal and anomalous events. Using the Faker library, the script produces random IP addresses, HTTP methods, status codes, API endpoints, and response times within a defined time range.

The process involves three main steps:

- **Timestamp creation:** Generating ordered timestamps across a specified period.

- **Anomaly interval definition:** Introducing intervals of abnormal behavior, such as client errors, server errors, and timeouts, by randomly selecting IP addresses and error types.
- **Dataset assembly:** Writing all generated events into a structured CSV file, including normal traffic and injected anomalies.

This automated generation ensures a diverse and controlled dataset that closely mimics real-world conditions, making it suitable for model training and evaluation.

Fig. 3. Logs generations

TABLE I
DESCRIPTION OF DATASET FEATURES

Column	Description
method	The HTTP request method used (e.g. GET, POST).
end_point	The targeted resource or URL (e.g. /login, /data).
response_time	The server's response time in milliseconds.
hour_of_day	The hour when the request was made (0–23).
day_of_week	The day of the week (0 = Monday, 6 = Sunday).
month	The month of the year (1–12).
is_weekend	Indicates whether the request occurred on a weekend (1 = yes, 0 = no).
index	The original log entry index in the dataset.
error_count	Number of errors (4xx or 5xx) for the IP address.
unique_error_types	Number of distinct error types encountered.
max_response_time	Average response time for the IP address.

C. Data Preprocessing

To prepare the log data for analysis and machine learning, a custom preprocessing pipeline was implemented. The process includes several key steps:

1) a. Feature Parsing and Transformation:

- Converted status_code to string format and extracted temporal features from the timestamp (hour, day of week, month).
- Added a binary indicator is_weekend to distinguish weekend traffic.
- Grouped logs by user_ip to calculate statistics such as error_count, unique_error_types, avg_response_time, and max_response_time.
- Created a categorical feature response_time_category (fast, normal, slow) based on response time.
- Added an is_potential_anomalous flag to mark potential anomalies (e.g., high latency or error codes).

2) b. Feature Engineering for Machine Learning:

- Selected numeric features (e.g., response times, error counts) to scale using StandardScaler.
- Selected categorical features (e.g., method, endpoint, weekend flag) to encode using OneHotEncoder.

3) c. Preprocessing Pipeline:

- Combined numeric and categorical transformations using ColumnTransformer.
- Ensured unknown categories during inference are handled gracefully.

4) d. Dataset Splitting:

- Implemented a function to split the dataset into training and testing subsets according to a configurable ratio.

This modular design ensures the dataset is properly cleaned, transformed, and ready for further anomaly detection and machine learning tasks.

D. Model Training

To detect abnormal behaviors in application logs, we implemented a machine learning-based anomaly detection module. Two unsupervised models were used:

1) Isolation Forest – an ensemble algorithm that isolates anomalies by randomly partitioning the data.
2) One-Class SVM (Support Vector Machine) – a boundary-based algorithm that identifies data points that deviate significantly from the normal profile.

The workflow is as follows:

- **Data preparation:** The preprocessed and engineered features obtained from the log dataset are split into training and testing sets.
- **Model training:** Both models are trained on the training data using the IsolationForest and OneClassSVM implementations from Scikit-learn.
- **Model persistence:** Trained models are saved to disk using joblib for later reuse without retraining.
- **Prediction and scoring:** For any new dataset, the system can predict whether each entry is normal or anomalous and compute anomaly scores for deeper analysis.

- **Evaluation:** The distribution of anomaly scores is visualized using histograms, and the ratio of detected outliers is calculated for each model.

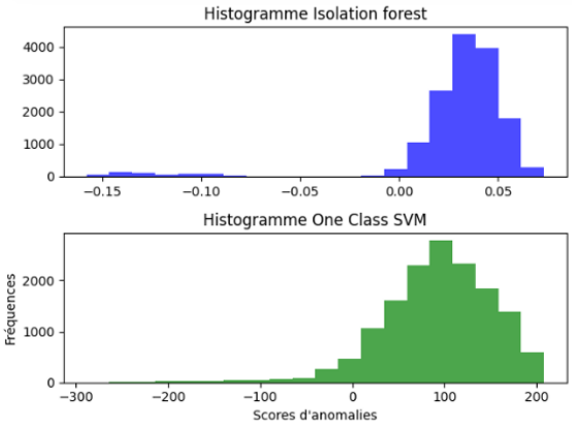

Fig. 4. Model Training

E. Results Interpretation

1) Top Histogram: Isolation Forest:

- **Description:** The histogram for the Isolation Forest algorithm is shown at the top.
- **X-axis:** Represents anomaly scores ranging from -0.15 to 0.05.
- **Y-axis:** Indicates the frequency of occurrences for each score.
- **Interpretation:** The majority of the data points are clustered near the 0.05 score, suggesting that the Isolation Forest algorithm has identified most of the data as normal. Scores close to zero indicate typical behavior, while lower scores may indicate detected anomalies.

2) Bottom Histogram: One-Class SVM:

- **Description:** The histogram for the One-Class SVM algorithm is displayed at the bottom.
- **X-axis:** Represents anomaly scores, with a wider range of values.
- **Y-axis:** Indicates the frequency of occurrences for each score.
- **Interpretation:** The One-Class SVM shows a different distribution of anomaly scores. The scores appear to be more spread out, indicating a wider variation in detected anomalies. This suggests that the One-Class SVM may have identified more outliers compared to the Isolation Forest, with some scores reflecting significant deviations from normal behavior.

F. API Development

In the API creation phase, I installed FastAPI and developed the `main.py` file to establish a robust framework for log anomaly detection. The FastAPI application is designed to handle various HTTP requests, providing endpoints for adding log entries, checking model statuses, and retrieving prediction results.

Utilizing Pydantic, I defined a LogEntry model to ensure data validation when logs are submitted. Redis was integrated as an in-memory database for efficient caching and session management, while Celery was employed to manage asynchronous tasks, enabling batch processing of log data.

Key functionalities include the ability to trigger anomaly detection when the log buffer exceeds a specified threshold, retrieve prediction results by their IDs, and monitor the status of processing tasks.

Fig. 5. Instalation of redis

Fig. 6. API Development

G. Log Simulation

A simulation process has been implemented to mimic the continuous flow of logs. The log simulator code reads entries from a CSV file containing log data and sends

each log to the previously designed API. This approach tests the model's ability to process and classify data streams in near real time.

Using the LogSimulator class, batches of logs are sent to the API, allowing for the evaluation of the anomaly detection system's effectiveness in a dynamic environment.

H. Deployment and Monitoring

Finally, the solution was deployed in a test environment with monitoring mechanisms in place to track model performance and adapt to changes in data patterns. The application generated status logs, indicating that the log processing and prediction tasks are functioning correctly.

Continuous monitoring ensures that the model remains effective and can be adjusted based on new trends detected in the data.

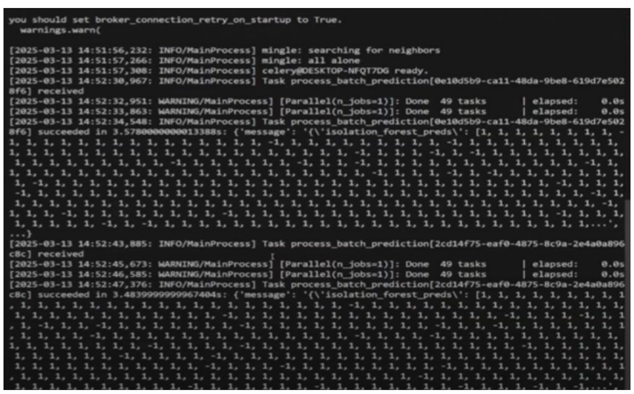

Fig. 7. Test environment

V. Conclusion

In conclusion, the development of a machine learning-based Intrusion Detection System (IDS) for malware detection represents a significant advancement in the ongoing battle against cyber threats. By leveraging powerful algorithms such as Random Forest and Support Vector Machine, this research not only enhances detection accuracy but also addresses the dynamic and evolving nature of malware.

The findings demonstrate that robust detection capabilities can be achieved, with high True Positive Rates and low False Positive Rates, ultimately enabling organizations to proactively safeguard their digital assets. As cyber threats continue to grow in complexity, integrating such intelligent detection systems into daily operations becomes imperative.

Future research should focus on refining these models and exploring innovative techniques that reduce reliance on human expertise in feature engineering, paving the way for more autonomous and adaptive cybersecurity solutions. This approach not only strengthens the resilience of networks but also contributes to the broader landscape of cybersecurity, fostering a more secure digital environment for all users.

References

[1] U.V. Nikam and V.M. Deshmukh, "Performance evaluation of machine learning classifiers in malware detection," in *Proceedings of the 2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, Ballari, India, 23–24 April 2022, pp. 1–5.

[2] K.V. Uma and E. Sharon Blessie, "Survey on Android malware detection and protection using data mining algorithms," in *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, IEEE, 2018, pp. 209–212.

[3] AAG IT Services, "The latest ransomware statistics," Available: https://aag-it.com/the-latest-ransomware-statistics/, Feb. 7, 2024.

[4] S. S. M. Rahman, F. B. Rafiq, T. R. Toma, S. Hossain, and K. Beepan Badruzzaman, "Performance assessment of multiple machine learning classifiers for detecting phishing URLs," in *Proceedings of 3rd ICDECT-2K19*, Springer, 2020, pp. 285–296.

[5] F. J. Aparicio-Navarro *et al.*, "Addressing multi-stage attacks using expert knowledge and contextual information," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management Workshops (IM)*, IEEE, 2019.

[6] F. J. Aparicio-Navarro *et al.*, "Multi-stage attack detection using contextual information," in *MILCOM 2018 - IEEE Military Communications Conference*, IEEE, 2018, pp. 1–9.

[7] M. F. Khan *et al.*, "Multi-stage attack detection: Emerging challenges for wireless networks," in *2022 International Conference on Smart Applications, Communications and Networking (SmartNets)*, IEEE, 2022.

[8] M. F. Khan *et al.*, "Latent Dirichlet allocation for the detection of multi-stage attacks," in *2023 International Conference on Information Technology*, IEEE, 2023.

[9] D. M. El Asaad *et al.*, "Anomaly detection using dynamic time warping," in *2019 IEEE International Conference on Computational Science and Engineering (CSE)*, IEEE, 2019, pp. 193–198.

[10] D. M. El Asaad *et al.*, "Denial of service detection using dynamic time warping," *International Journal of Network Management*, vol. 31, no. 6, 2021.

[11] M. Hammad *et al.*, "A service-oriented approach for sensing in the Internet of Things," *IEEE Sensors Journal*, vol. 14, no. 11, 2020, pp. 15753–15761.

[12] M. Hammoud and R. Newman, "Information extraction from sensor networks using the Watershed transform algorithm," *Information Fusion*, vol. 22, 2015, pp. 309–319.

[13] S. Kumar *et al.*, "MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things," *Future Generation Computer Systems*, vol. 125, 2021, pp. 334–351.

[14] S. M. Lima *et al.*, "Artificial intelligence-based antivirus in order to detect malware preventively," *Progress in Artificial Intelligence*, vol. 10, no. 1, 2021.