Available online at www.qu.edu.iq/journalcm
JOURNAL OF AL-QADISIYAH FOR COMPUTER SCIENCE AND MATHEMATICS
ISSN:2521-3504(online) ISSN:2074-0204(print)

Malware classification using SVM and XGBoost: A study of the influence of features

Mohamed Fadhil Imran

Computer Engineering Department, Faculty of Engineering, Islamic Azad University, Tehran Branch, Iran.

.Email:mohamademrany1991@gmail.com

ARTICLE INFO

Article history:
Received: 02 /08/2025
Rrevised form: 24 /08/2025
Accepted : 26 /08/2025
Available online: 30 /12/2025

Keywords:

Malware
Zero-day
machine learning
SVM
XGBoost
Meta-features

ABSTRACT

Zero-day malware shows a significant cybersecurity threat due to its ability to avoid traditional antivirus detection. This study investigates the application of machine learning techniques to detect and classify such advanced threats. Specifically, two well-known machine learning classifiers: Support Vector Machine (SVM) and XGBoost, to classify malware in a static analysis environment. The data was collected from the Kaggle platform and includes 5212 samples containing 70 features extracted from PE (Portable Executable) headers, including features from the DOS Header, PE Header, and Optional Header. The samples include both malicious and healthy files, and ten-fold cross-validation was used. The performance of the classifiers was evaluated using metrics such as accuracy, recall, F1 score, Roc curve and positive precision.

The results demonstrate XGBoost's performance was superior with an accuracy of about 99%, while SVM's accuracy was about 96%. XGBoost shows its high effectiveness and low error rate, which enhances its potential to be used as an effective tool in detecting zero-day malware and improving cyber security capabilities to detect unknown threats.

MSC.

https://doi.org/10.29304/jqcsm.2025.17.42536

1. Introduction

The Internet of Things (IoT) is one of the most prominent recent technological developments, providing advanced capabilities in remote data monitoring and supporting automated decision-making processes [1][2].

However, with the significant expansion in the use of IoT devices and their increasing reliance on cloud computing environments, major security challenges have emerged, including unauthorized access, sensitive data leaks, and other complex cyber threats [3][4].

∗Corresponding author

Email addresses:

Communicated by ‘sub etitor’

Traditional intrusion detection methods such as signature-based detection and heuristic analysis have been relied upon [5], but these methods have become insufficient to counter modern threats, especially with the use of techniques such as obfuscation, polymorphic forms, and exploitation of unknown vulnerabilities.

As a result, there is a need for more intelligent and adaptive methods of attack detection, and machine learning (ML) techniques have emerged as a powerful tool due to their ability to recognize unusual patterns and new threats that traditional systems may fail to detect [6][7][8]. These systems have significantly outperformed the latest available methods, achieving advanced performance that enhances the cyber security capabilities of IoT networks [9].

ML models demonstrate a unique ability to automatically extract features from raw data, reducing the need for human intervention and improving scalability. They have also proven effective in handling complex data such as byte sequences and API calls, leading to improved classification accuracy in malware detection tasks [10].

However, the effectiveness of these systems depends heavily on the availability of high-quality training data, which is a fundamental challenge in complex IoT environments, where comprehensive and sufficiently representative data is always difficult to obtain [11][12].

The static analysis approach has been adopted for extracting malware characteristics, as it provides a safe representation of software behavior without the need to execute it, thereby reducing operational risks and making it a suitable option in resource-constrained environments.

This method has proven effective in machine learning-based detection systems, with a recent system for studying malware using static analysis achieving a detection rate of 99.5% and a low error rate of less than 0.47%, demonstrating its high accuracy and classification efficiency [13].

Many static malware detection approaches have been characterized by their examination of various classification methods, including support vector machines (SVM), k-nearest neighbor (KNN) algorithms, Naive Bayes (NB) algorithms, and others. According to the results, XGBoost classification can achieve an accuracy of over 90% [14].

A study [15]. showed that the XGBoost algorithm achieved high performance using static analysis on Meraz'18 data, competing with the most common classification algorithms in this field, which reinforces its importance as a leading candidate for comparison with the SVM algorithm in this study. Similar study used memory analysis techniques to extract features, using the SVM algorithm, which achieved a high accuracy of 98.5% [16].

An antivirus tool known as MLMD was developed [17], based on an XGBoost machine learning algorithm for classifying malware. The researchers trained models using the XGBoost and Extra Trees algorithms on two datasets obtained through various analyses of malicious and benign samples. They then compared their performance with several other classification algorithms, including Random Forest, Decision Tree, SVM and Naive Bayes. The results showed that the best performance was achieved using the XGBoost algorithm, which recorded an accuracy of 91.9% on the fixed analysis data, a sensitivity of 98.2%, an area under the ROC curve (AUC) of 0.853, and an F1 score of 0.949.

This study highlights the effectiveness of the XGBoost algorithm in static analysis and its superiority over well-known algorithms such as SVM and Random Forest, which highlights the importance of a direct comparison between XGBoost and SVM, which is the aim of this research.

Despite these indicators, most studies in this field refer to comprehensive comparisons between many classification algorithms, without focusing on a direct binary comparison between two specific algorithms [18], which highlights the research gap that this study seeks to address.

The main objective of this study is to conduct an analytical comparison between two popular algorithms in this field for classifying malware using static analysis and to evaluate their performance based on a set of standard metrics.

This study is particularly important because it relies on static analysis, which is a suitable option in secure environments, as it allows software to be analyzed without executing it, thereby reducing operational risks. Furthermore, the comparison between two popular algorithms in the field of machine learning helps guide researchers and developers towards more effective and accurate techniques.

Given the existence of more popular algorithms, such as Random Forest, which has achieved the best accuracy rates in many studies [19], they were excluded from our study to focus on less commonly used algorithms, such as XGBoost and SVM, which show similar performance, allowing for a deeper evaluation of their effectiveness within the framework of static analysis.

Another recent study [20] pointed out that machine learning–based malware detection systems face practical security risks that go beyond accuracy. Using a stage-wise classification grounded in the CIA principles, they provided a framework for analyzing vulnerabilities at each stage, supported by case studies that offer practical insights into both defensive and offensive challenges. This complements our study, which focused on improving the performance of SVM and XGBoost with meta-features and highlights the importance of considering model security.

Based on the above, this study focused on the SVM and XGBoost algorithms to analyze the impact of meta-features on malware classification performance. Although the Random Forest algorithm may provide high accuracy, its complex nature as an ensemble of trees makes it difficult to clearly interpret the contribution of each feature, especially after adding the new features. Therefore, it was temporarily excluded to ensure clarity in the analysis and to facilitate the evaluation of the impact of meta-features on the selected models.

2. Methodology

The proposed approach shows the process of comparing SVM and XGBoost classifiers for malware classification. Their performance is evaluated with calculation several metrics such as accuracy, precision etc. The experiments were conducted in two phases:

The experiments were conducted in two phases: the first using only the original feature set, and the second after enhancing the dataset by adding meta-features related to file characteristics such as entropy, file size, imports count, and the presence of suspicious API calls.

This experiment aimed to measure the impact of these additional features on improving the performance of the models and to determine the extent to which each algorithm benefits from them. Our methodology stages are shown in **Figure 1. Methodology Steps.**

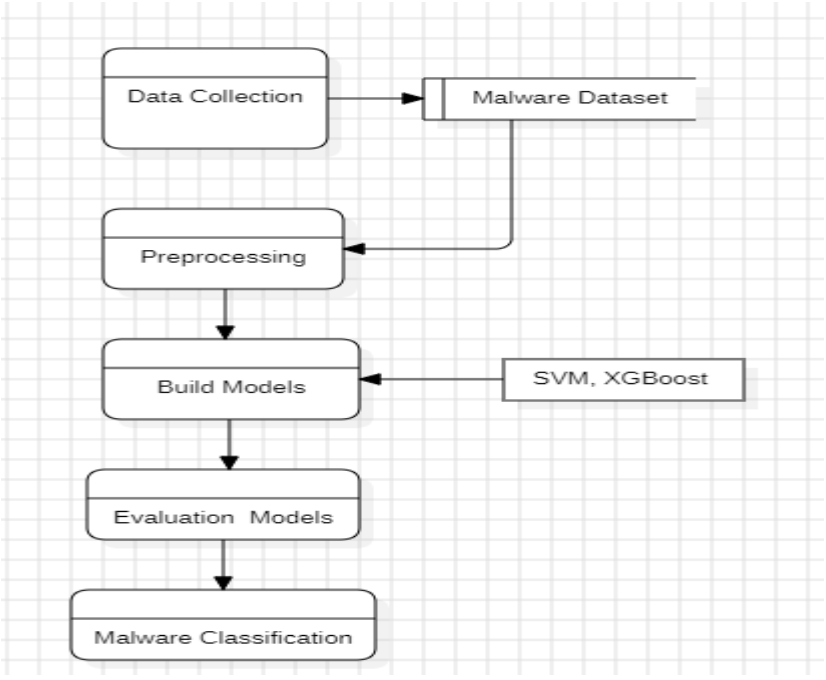

FIGURE 1. METHODOLOGY STEPS.

2.1 Data Collection

The dataset used in our study was collected from Kaggle and includes 5210 executable file samples with each sample described by 70 features, that indicating a slightly imbalanced distribution. In addition, a set of meta-features (such as entropy values, file size, number of sections, and section size ratios) was derived to enhance the performance of the models. These samples represent both malicious and benign portable executable files extracted using static analysis methods [22]. The distribution of classes based on this dataset:

- 722 malware samples represent with class feature equal 1
- 2488 benign samples represent with class feature equal 0.

	e_cblp	e_cp	e_cparhdr	e_maxalloc	e_sp	e_lfanew	NumberOfSections	CreationYear	FH_char0	FH_char1	...	sus_sections	non_sus_sections	packer	packer_type	E_text	E_data	filesize	E_file	fileinfo	class
2488	144.0	3.0	4.0	65535.0	184.0	240.0	4.0	1.0	1.0	1.0	...	0.0	4.0	0.0	NoPacker	7.613436	7.874120	129024.0	7.461757	1.0	1.0
2489	144.0	3.0	4.0	65535.0	184.0	232.0	6.0	1.0	1.0	1.0	...	2.0	4.0	0.0	NoPacker	6.083758	0.814913	18432.0	5.433045	0.0	1.0
2490	144.0	3.0	4.0	65535.0	184.0	224.0	5.0	1.0	0.0	1.0	...	2.0	3.0	0.0	NoPacker	7.951998	6.428710	201151.0	7.393284	1.0	1.0
2491	80.0	2.0	4.0	65535.0	184.0	124.0	4.0	1.0	1.0	1.0	...	0.0	4.0	0.0	NoPacker	5.313268	0.100515	189952.0	7.876330	0.0	1.0
2492	144.0	3.0	4.0	65535.0	184.0	184.0	4.0	1.0	1.0	1.0	...	1.0	3.0	0.0	NoPacker	7.815046	0.000000	108544.0	7.878427	0.0	1.0
0	144.0	3.0	4.0	65535.0	184.0	256.0	4.0	1.0	0.0	1.0	...	1.0	3.0	0.0	NoPacker	6.603616	5.443362	1181520.0	6.627552	1.0	0.0
1	144.0	3.0	4.0	65535.0	184.0	184.0	4.0	1.0	0.0	1.0	...	1.0	3.0	0.0	NoPacker	5.205926	2.123522	7680.0	5.318221	0.0	0.0
2	144.0	3.0	4.0	65535.0	184.0	272.0	5.0	1.0	0.0	1.0	...	1.0	4.0	0.0	NoPacker	6.238000	3.380859	57872.0	6.507758	1.0	0.0
3	144.0	3.0	4.0	65535.0	184.0	184.0	1.0	1.0	0.0	1.0	...	0.0	1.0	0.0	NoPacker	0.000000	0.000000	95616.0	4.575092	1.0	0.0
4	144.0	3.0	4.0	65535.0	184.0	224.0	5.0	1.0	0.0	1.0	...	1.0	4.0	0.0	NoPacker	6.355626	0.702621	48128.0	5.545531	1.0	0.0

FIGURE 2. DATASET CONTENT.

According to **Figure 2**, the dataset consists static attributes extracted from Windows PE files. These features can be grouped as follows:

- DOS header fields are basic legacy header fields from MS-DOS, sometimes manipulated by packers or obfuscation tools.
- File header flags are bitwise flags from the PE File Header, which indicate characteristics like DLL type, debug info, and subsystem.

- Metadata about the compiler/linker and layout of code/data in memory.
- Memory allocation and structure fields. These are relevant for analyzing how the binary behaves during execution.
- Information about operating system and subsystem versions targeted by the executable.
- Flags from the optional header, which indicate options related to security like ASLR, DEP, etc.
- Control memory usage by the program that may be manipulated by malicious code.
- Additional analysis of whether the binary is packed or has suspicious sections.
- Entropy features measure randomness, high entropy often indicates encryption/obfuscation.
- The target column is label, which include target variable 1 indicates to malware and 0 to benign.

2.2 Preprocessing

To ensure the dataset was ready for training, and enhance performance of model the following preprocessing steps were applied:

- Convert categorical columns into numerical format.
- Handling missing values: Missing values were addressed by removing rows containing missing values in the target column, while missing values in numerical features were imputed using the mean of each column to ensure that the models received complete data ready for training.
- Feature Scaling: Numerical features were scaled using StandardScaler, which standardizes each feature to have a mean of 0 and a standard deviation of 1, ensuring uniformity across features and improving the performance of algorithms sensitive to feature magnitudes, such as SVM.
- Feature scaling, numerical features were scaled to ensure uniformity for algorithms sensitive to scale.
- The models were evaluated using 10-Fold Cross Validation, ensuring stratified splits to preserve class balance in each fold.
- As noted in the introduction to the chapter, the data were later augmented by adding meta-features based on technical characteristics of the executable, to improve the efficiency of the models.

2.3 Build Models

A. SVM

SVM is a supervised machine learning algorithm used for binary classification tasks. It works by finding the optimal hyperplane that classifies the classes into malware or benign with the maximum margin. It's particularly effective in high-dimensional spaces and works well for both linear and non-linear classification using kernel tricks. In Table 1.

TABLE 1. HYPERPARAMETER FOR SVM.

Hyperparameter	***Value***	***Role***
Kernel	Rbf	Allows SVM to classify non-linear data.
C	1.0	Controls the trade-off between a smooth decision boundary and classifying training points correctly.
Gamma	Scale	Defines how far the influence of a single training example reaches.
Class weight	Balanced	Automatically adjusts weights inversely proportional to class frequencies.
Probability	True	Enables probability estimates, which required for ROC/AUC evaluation.
Random state	42	Sets seed for reproducibility.

Based on **Table 1**, The RBF kernel was chosen because it excels in malware classification, being able to capture complex nonlinear patterns. It also provides greater flexibility and adaptability to the data shape through the gamma parameter, often achieving strong performance across a wide range of datasets, making it a better choice compared to linear and polynomial kernels.

B. XGBoost

XGBoost is a powerful gradient boosting algorithm that builds an ensemble of decision trees sequentially. Each tree corrects the errors of the previous ones by allowing the model to learn complex patterns and achieve high

performance with efficient training. It's well-known for its speed and accuracy in tabular data classification tasks like malware detection. The hyperparameter appears in **Table 2.خطأ! لم يتم العثور على مصدر المرجع**.

TABLE 2. HYPERPARAMETER FOR XGBOOST.

Hyperparameter	*Value*	*Role*
N_estimators	300	The number of trees the model builds, more tree high performance.
Learning_rate	0.1	Step size shrinkage used in updates to prevent overfitting.
Max_depth	6	Maximum depth of each decision tree, to capture more patterns but may overfit.
Subsample	0.8	Fraction of training data to be used for each boosting round, to prevent overfitting and speeds up training.
Colsample_bytree	0.8	Fraction of features used for constructing each tree.
Random state	42	Seed for reproducibility.

According to **Table 2.خطأ! لم يتم العثور على مصدر المرجع**, The choice of max_depth=6 was made to strike a balance between model complexity and performance, ensuring that the model remains robust, efficient, and capable of learning the most important patterns.

2.4 Evaluation Models

After training the SVM and XGBoost models, their performance was evaluated using various of classification metrics are defined as below [21]:

- Accuracy is the ratio of correctly predicted samples both malware and benign to the total number of samples. It provides insight how often the model is right overall but may be misleading in imbalanced datasets if most files are benign.
- Precision is the ratio of correctly predicted malware files to all files the model predicted as malware. High precision means few false alarms, so most files flagged as malware really are malware.
- Recall is the ratio of correctly predicted malware files to all actual malware files. High recall means the model catches most of the malware, even if it occasionally flags benign files incorrectly. It's important to avoid letting malware go undetected.
- F1 Score is the harmonic mean of precision and recall. It balances both false positives and false negatives, which a very useful metric, especially when need to balance detection rate (recall) with avoiding false alarms (precision).
- Confusion matrix is crucial to understanding the exact types of errors model makes.
- ROC curve plots true positive rate vs. false positive rate at different classification thresholds, where high AUC means the model can reliably separate malware from non-malware, even under different threshold settings.

2.5 Malware Classification

The final step involves binary classification where:

- Output 1 means malware
- Output 0 means benign

Predictions are evaluated using the test set, and confusion matrices were generated to visually inspect classification performance.

3. Results and Discussion

3.1 Results of SVM Model

A. Confusion Matrix

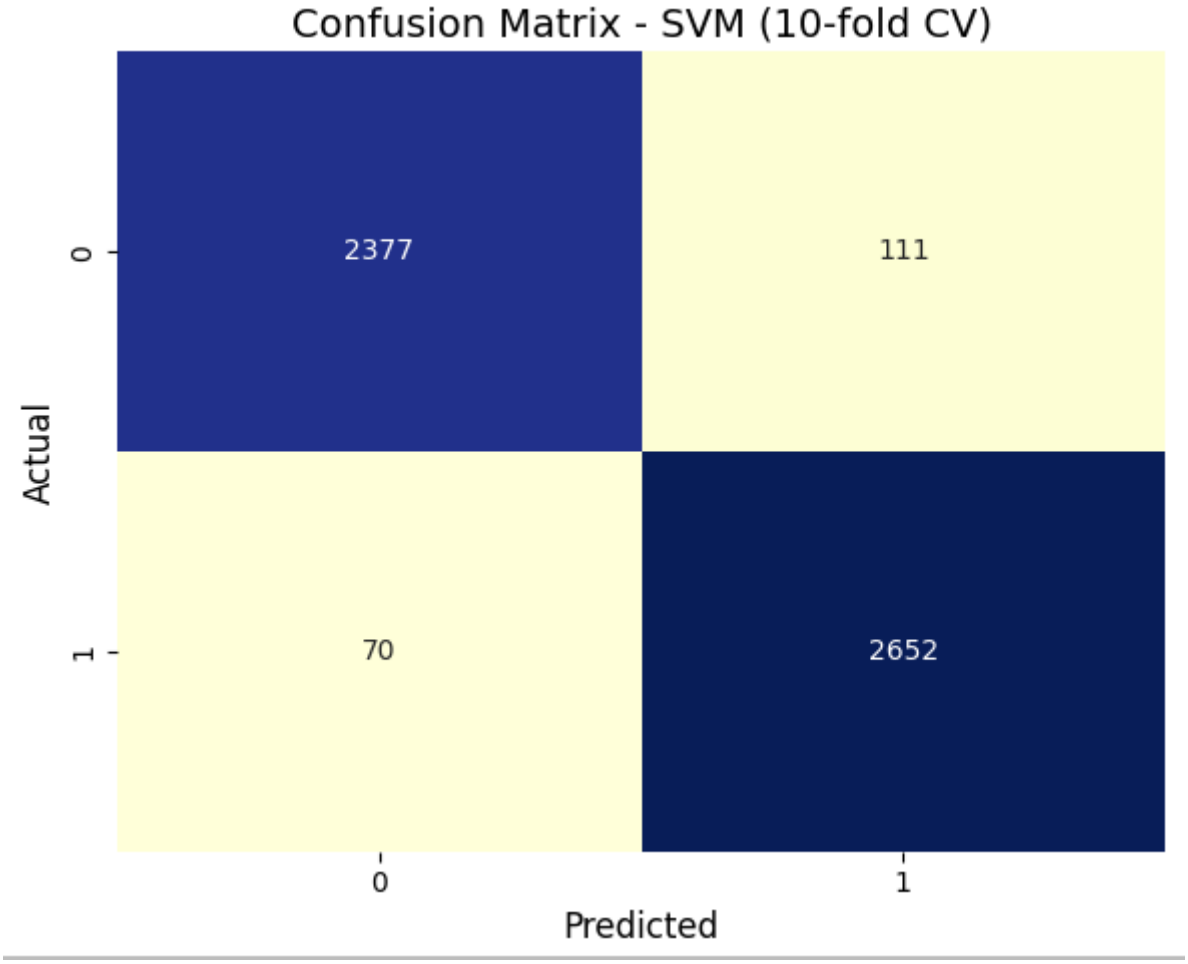

FIGURE 3. CONFUSION MATRIX FOR SVM.

As shown in **Figure 3**, after using the 10-fold cross-validation method, the classification results were represented in the confusion matrix shown in Figure (3), where the model showed a clear ability to distinguish between benign and malicious software. A total of 2,652 malicious cases and 2,377 benign cases were classified correctly, while 111 benign cases were misclassified as malicious and 70 malicious cases were misclassified as benign.

B. Summary of Classification

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.96	0.96	2488
1	0.96	0.97	0.97	2722
accuracy			0.97	5210
macro avg	0.97	0.96	0.97	5210
weighted avg	0.97	0.97	0.97	5210

FIGURE 4. CLASSIFICATION REPORT FOR SVM.

Based on **Figure 4.خطأ! لم يتم العثور على مصدر المرجع**, The model's accuracy was 97%, indicating that 97% of the samples were classified correctly. Both the macro average and weighted average showed a value of 0.97, reflecting a good balance in the model's performance between the two categories: benign and malicious software. Despite the similarity of the indicators between the two categories, the model showed a slight bias towards the positive category (malicious software), where the recall value was higher (0.97 vs. 0.96 for the benign category). This represents the algorithm's outstanding ability to detect malicious software.

C. Roc Curve

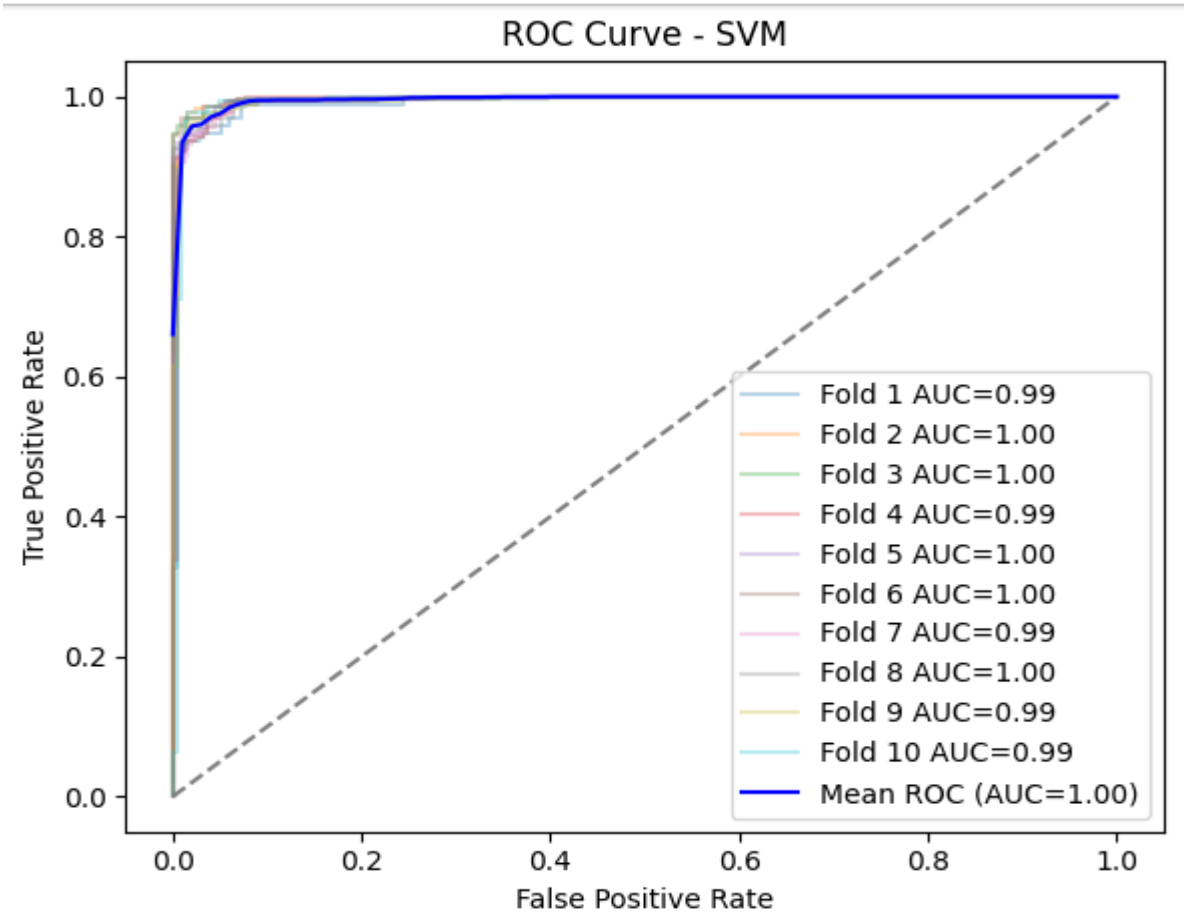

FIGURE 5. ROC CURVE FOR SVM.

After analysis area under the ROC curve across all folds for SVM model, it scores ranged between 0.90 and 1.00 as shown in **Figure 5**. This difference indicates that while the model consistently performed at a high level across all folds, some subsets of the data posed slightly greater classification challenges, likely due to overlapping characteristics between malware and benign samples. Folds with an AUC of 1.00 indicate to perfect separability, where the model made no classification errors whereas folds with an AUC of 0.90 still showed excellent discriminatory power, though with minimal misclassifications. In summary, these results confirm the robustness and reliability of the SVM model in detecting malware from static PE features, maintaining strong generalization across different data partitions.

Although the obtained AUC values were very high (close to 1.0), this does not necessarily indicate over fitting. Several measures were taken to mitigate such risks, including the use of cross-validation to ensure result stability across different subsets of the data, as well as applying data preprocessing steps to minimize the chances of data leakage. Moreover, the nature of the features used (PE header features) reflects real characteristics associated with malware, which further enhances the reliability of the results. Nonetheless, such extremely high values could still

indicate a degree of bias or over fitting, especially if the dataset size is limited or lacks diversity. Therefore, it is recommended that the models be tested in the future on larger and more diverse datasets.

3.2 Results of XGBoost Model

a) Confusion Matrix

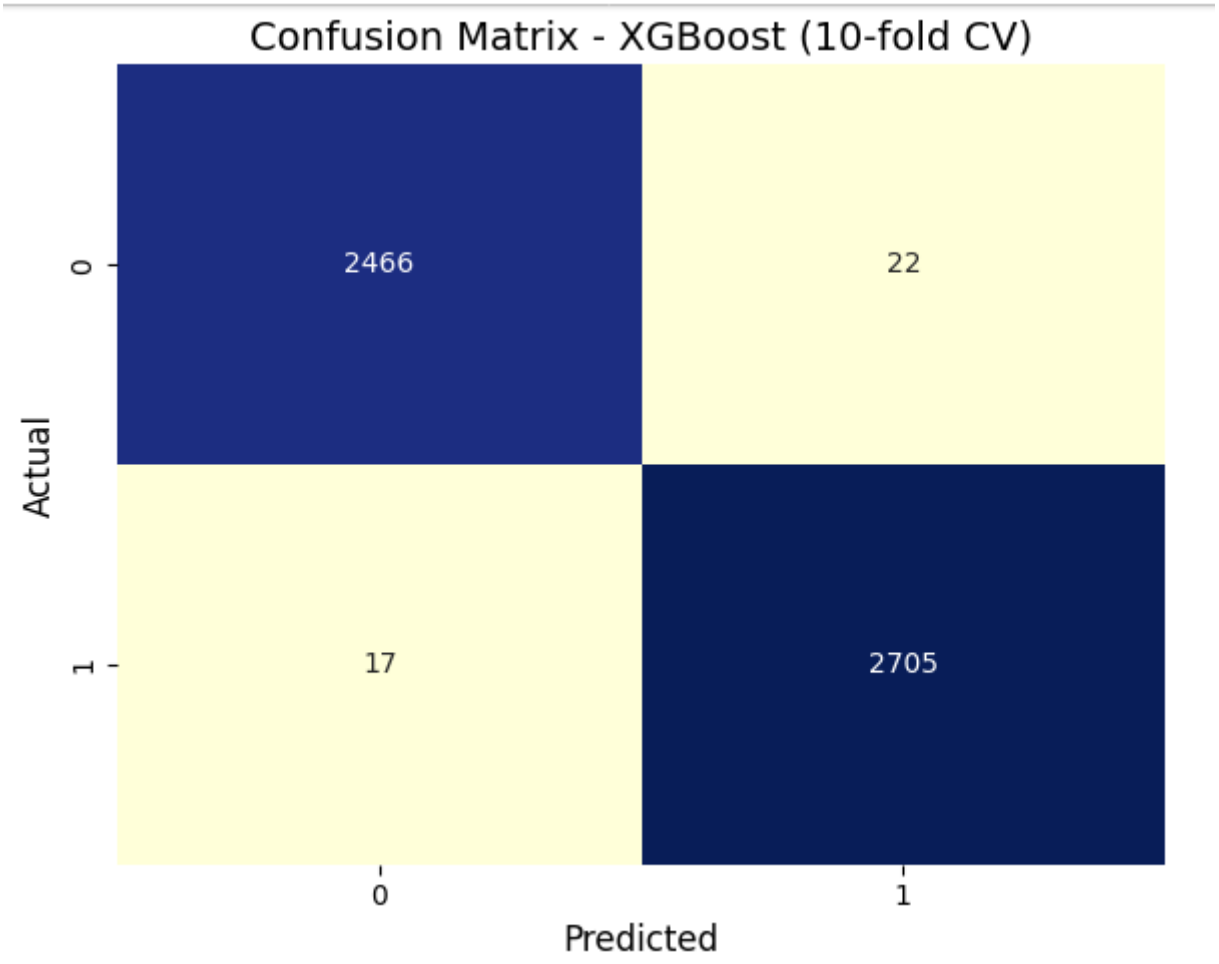

FIGURE 6. CONFUSION MATRIX FOR XGBOOST.

According to **Figure 6.خطأ! لم يتم العثور على مصدر المرجع**, after using the 10-fold cross-validation method, the classification results were represented in the confusion matrix shown in Figure 4, where the model showed excellent performance in distinguishing between benign and malicious software. A total of 2,705 malicious cases and 2,464 benign cases were classified correctly, while 22 benign cases were misclassified as malicious and 17 malicious cases were misclassified as benign.

b) Summary of Classification

Based on **Figure 7**, the model's accuracy reached 99%, indicating that it was able to correctly classify 99% of the samples into two categories (legitimate and malicious software). The macro average and weighted average indicators also showed the same value (0.99) for precision, recall, and F1 score, reflecting the model's balanced and robust performance across both categories. Although the model's performance was nearly identical between the

benign (0) and malicious (1) categories, these results demonstrate an exceptional ability to detect malicious software without affecting the classification of benign software, which is critical in the context of cyber security.

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	2488
1	0.99	0.99	0.99	2722
accuracy			0.99	5210
macro avg	0.99	0.99	0.99	5210
weighted avg	0.99	0.99	0.99	5210

FIGURE 7. CLASSIFICATION REPORT FOR XGBOOST.

c) Roc Curve

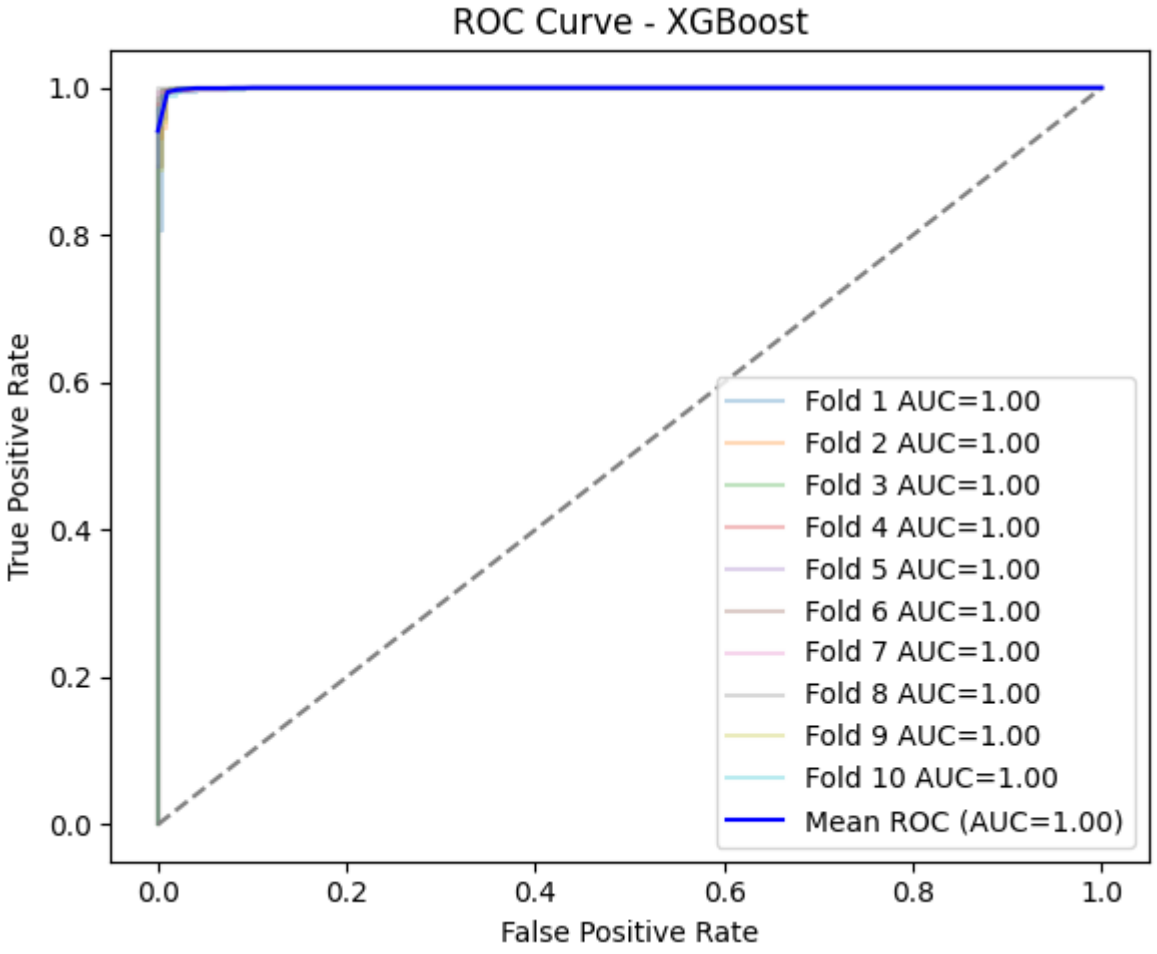

FIGURE 8. ROC CURVE FOR XGBOOST.

As shown in **Figure 8**, the XGBoost model achieved an AUC score of 1.00 across all 10 cross-validation folds, which indicate to perfect classification performance in every subset of the data. This exceptional result shows that the model was able to flawlessly distinguish between malware and benign files with no false positives or false negatives in any fold. Such consistency appears XGBoost's ability to capture complex patterns within the static PE

header features used in this study. In summary, the model's high reliable for detecting even subtle forms of malware in real-world scenarios.

3.3 Comparing Between Models

According to **Figure 9**, both the SVM and XGBoost models showed high effectiveness in detecting malware using static PE header features. However, a closer look at the evaluation metrics reveals that XGBoost consistently outperformed SVM across all key indicators.

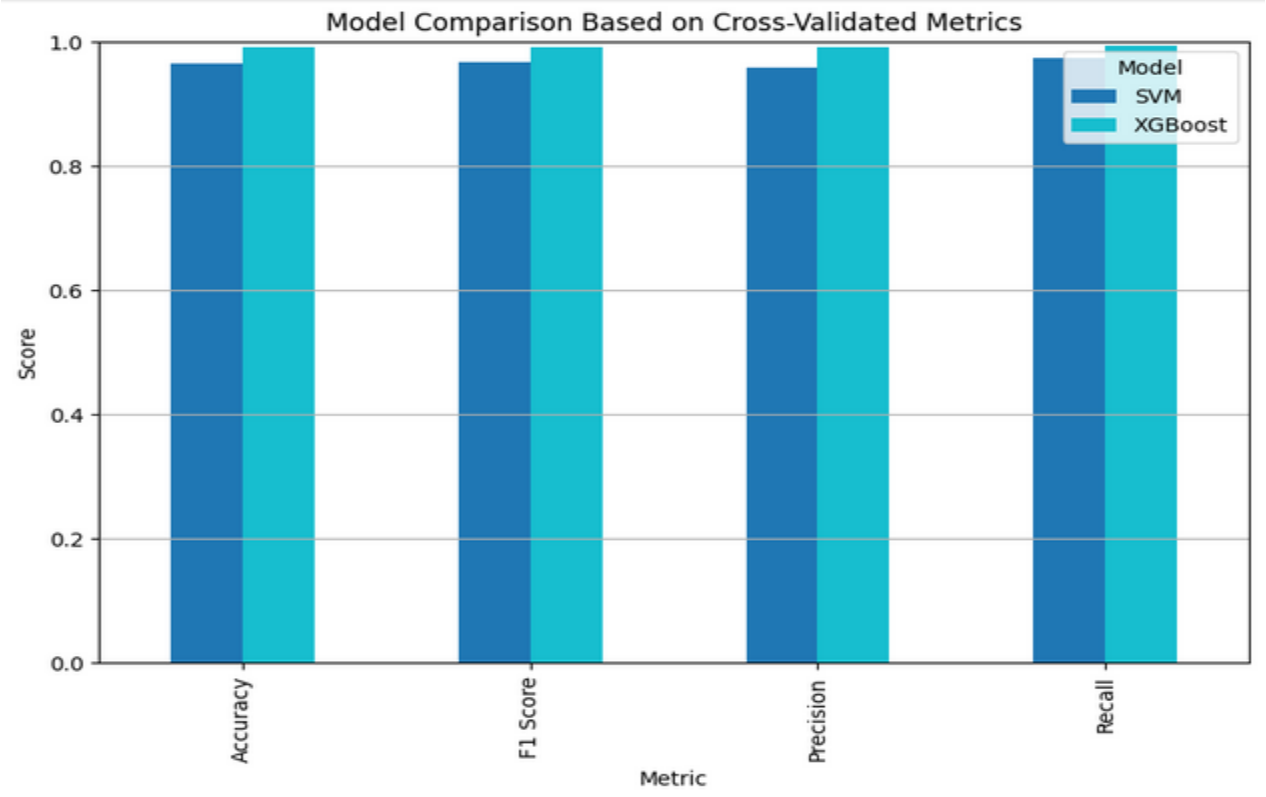

FIGURE 9. COMPARISON BETWEEN MODELS.

Comparison between our models based on classification metrics is shown as blow:

- XGBoost accomplished an accuracy of 99% compared to 97% for SVM, that made it fewer overall classification errors.
- XGBoost had 0.99 precision for both classes that meaning it almost never falsely identified benign files as malware or vice versa. SVM's precision was slightly lower (0.97 for class 0 and 0.96 for class 1), which indicates to a slightly higher false positive rate.
- XGBoost also achieved a recall of 0.99 for both classes reflecting its superior ability to correctly identify actual malware and benign samples. SVM's recall was 0.96 for class 0 and 0.97 for class 1, which suggests a slightly higher false negative rate for benign files.
- XGBoost safeguarded a balanced F1-score of 0.99, appearing excellent harmonic means between precision and recall, while SVM's F1-score ranged between 0.96 and 0.97, still strong but comparatively lower.

In summary, both models are highly capable, XGBoost showed superior classification performance in all evaluated metrics, by making it a more reliable and accurate choice for malware detection in this static analysis context.

3.4 Comparing models after adding meta features

After completing the initial comparison between the SVM and XGBoost classification algorithms using the original feature set and obtaining the baseline values for the performance indicators and confusion matrix for each model, we moved to the next stage of analysis.

In this stage, the Meta features were combined with the original data, and the same models were then retrained on this expanded dataset to improve performance and reduce error rates.

This step aimed to evaluate the impact of the additional features on the model results, as well as to determine which of the two algorithms benefited most from the inclusion of these new features.

TABLE 3. COMPARING CLASSIFICATION RESULTS BEFORE AND AFTER APPLYING META FEATURES TO THE MODEL.

Standard	SVM before	SVM after	XGBoost before	XGBoost after
Accuracy	96.53%	99.06%	99.21%	99.23%
Precision	95.99%	98.80%	99.12%	99.27%
Recall	97.43%	99.41%	99.38%	99.27%
F1-score	96.70%	99.10%	99.25%	99.27%
TN	2337	2455	2466	2468
FP	111	33	22	20
FN	70	16	17	20
TP	2652	2706	2705	2702

This **Table 3**, shows that the SVM algorithm's performance improved significantly after adding meta-features, with a significant reduction in classification errors, leading to a significant increase in the model's efficiency and ability to distinguish between malicious and non-malicious software. In contrast, the XGBoost algorithm maintained its high performance with minor changes in some performance metrics. This analysis reflects that adding meta-features had a significant impact on improving SVM performance. Figure 10 shows the difference in the performance of SVM and XGBoost algorithms after adding Meta features, by comparing four basic metrics: Accuracy, Precision, and Recall.

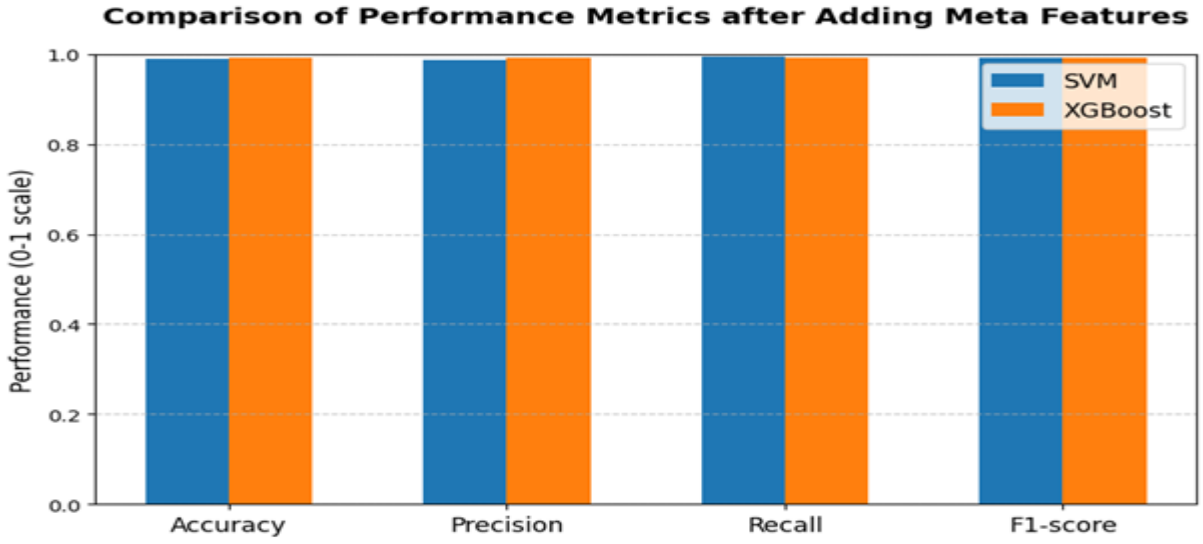

FIGURE 10. COMPARISON OF MODELS AFTER ADDING META FUTURE FEATURES.

The **Figure 10** shows that the SVM algorithm's performance improved significantly after adding meta-features, with accuracy increasing from 96.53% to 99.06%, positive precision increasing from 95.99% to 98.80%, recall increasing from 97.43% to 99.41%, and the F1 index increasing from 96.70% to 99.10%. In contrast, the XGBoost algorithm maintained its high performance, with slight improvements in most metrics, with accuracy increasing from 99.21% to 99.23% and positive precision from 99.12% to 99.27%. Recall, however, decreased from 99.38% to 99.27%, and the F1 index increased from 99.21% to 99.27%. This analysis shows that the impact of adding meta-features was more pronounced on SVM performance, bringing its results very close to those of XGBoost, and significantly reducing the gap between the two models.

4. Conclusion

This paper provided a deep comprehensive analytical comparison between the SVM and XGBoost algorithms for malware classification Before and after the addition of Meta-features using static analysis of PE header features, The XGBoost showed the clear superiority before adding features, which achieved an accuracy of 99.21% compared to 96.53% for SVM. XGBoost outperformed SVM across all key evaluation metrics. These results highlight the reliability and higher classification efficiency of XGBoost, making it a highly effective model for static malware detection, especially in IoT-based cybersecurity applications.

Based on area under Roc curve, XGBoost has AUC score of 1.00 across all folds, which emphasizes its excellent ability to distinguish between malware and benign files, while the SVM model showed slight variation with AUC scores ranging from 0.90 to 1.00 that still reflecting high but less consistent performance.

While XGBoost demonstrated a clear superiority in the original analysis before adding meta-features, the SVM algorithm had a different opinion after introducing these features, with its accuracy increasing from 96.53% to 99.06%. Other performance metrics also increased, and the error rate decreased, making it a strong competitor, achieving high performance and approaching XGBoost, which maintained its high performance. These results reflect the importance of using meta-features to improve the capabilities of machine learning models, especially in malware classification, where additional features can enhance models' ability to more accurately distinguish between malicious and benign patterns.

Based on the findings, the following recommendations are proposed:

- Implement XGBoost in real world scenarios as the core classifier in static-analysis-based malware detection systems for its superior accuracy and precision.
- Enhance classification performance by using deep learning approaches such as Convolutional Neural Networks (CNN).
- Use other datasets include a broader range of malware types especially zero-day threats to improve generalization.
- Integrated between static and dynamic analysis techniques to capture both code structure and runtime behavior that way reducing false positives and improving detection robustness.

In conclusion, this study shows the importance of selecting high performance algorithms like XGBoost for malware detection in critical systems and encourages future research focused on combining multiple techniques to address evolving cybersecurity threats in real-world environments.

References

[1] M. Nassereddine and A. Khang, Applications of Internet of Things (IoT) in Smart Cities, in Advanced IoT Technologies and Applications in the Industry 4.0 Digital Economy, CRC Press, 2024, pp. 109–136.

[2] M. Hossain, G. Kayas, R. Hasan, A. Skjellum, S. Noor, and S. R. Islam, "A holistic analysis of Internet of Things (IoT) security: Principles, practices, and new perspectives," Future Internet, vol. 16, no. 2, p. 40, 2024.

[3] Z. Azam, M. M. Islam, and M. N. Huda, "Comparative analysis of intrusion detection systems and machine learning-based model analysis through decision tree," IEEE Access, 2023.

[4] Z. Chen et al., "A survey on security threats and countermeasures in the Internet of Things," ACM Computing Surveys, Apr. 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.03433

[5] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," IEEE Access, vol. 8, pp. 6249–6271, 2020, doi: 10.1109/ACCESS.2019.2963724.

[6] H. Rathore, S. Agarwal, S. K. Sahay, and M. Sewak, "Malware detection using machine learning and deep learning," in Big Data Analytics, A. Mondal, H. Gupta, J. Srivastava, P. Reddy, and D. Somayajulu, Eds., Cham: Springer, 2018, pp. 402–411, doi: 10.1007/978-3-030-04780-1_28.

[7] S. V. N. Santhosh Kumar, M. Selvi, and A. Kannan, "A comprehensive survey on machine learning-based intrusion detection systems for secure communication in Internet of Things," Computational Intelligence and Neuroscience, vol. 2023, p. 8981988, 2023.

[8] V. Kukartsev et al., "Using machine learning techniques to simulate network intrusion detection," in 2024 Int. Conf. on Intelligent Systems for Cybersecurity (ISCS), 2024, pp. 1–4.

[9] M. A. Amer and R. Svyd, "Intelligent cyber-attack detection in IoT networks using IDAOA-based wrapper feature selection," Wasit Journal for Pure Science, Jun. 2025. [Online]. Available: https://doi.org/10.31185/wjps.731

[10] C. Bugra and D. Erdogan, "Malware classification using deep learning methods," in Proc. ACM Southeast Conf. (ACMSE '18), 2018, pp. 1–5, doi: 10.1145/3190645.3190692.
[11] I. Hidayat, M. Z. Ali, and A. Arshad, "Machine learning-based intrusion detection system: an experimental comparison," Journal of Computational and Cognitive Engineering, vol. 2, no. 2, pp. 88–97, 2023.
[12] R. Saadouni et al., "Intrusion detection systems for IoT based on bio-inspired and machine learning techniques: A systematic review of the literature," Cluster Computing, pp. 1–27, 2024.
[13] A. Yousuf et al., "Windows malware detection based on static analysis with multiple features," arXiv preprint, arXiv:2304.03433, 2023. [Online]. Available: https://arxiv.org/abs/2304.03433
[14] F. Sareen, Suzaifa, and A. Khader, "Comparative analysis of malware classification using machine learning algorithms," Int. J. of Engineering Research and Applications, vol. 10, no. 12, 2020.
[15] A. F. A. Arizal, M. Md-Arshad, A. Abdul-Samad, M. Md Sirat, and S. H. Othman, "Performance comparative study on zero-day malware detection using XGBoost and Random Forest classifiers," Int. J. of Innovative Computing, vol. 14, no. 2, Nov. 2024. [Online]. Available: https://doi.org/10.11113/ijic.v14n2.449
[16] R. Sihwail, K. Omar, and K. A. Z. Ariffin, "An effective memory analysis for malware detection and classification," Comput. Mater. Contin., vol. 67, no. 2, pp. 2301–2320, 2021, doi: 10.32604/cmc.2021.014510.
[17] J. Palša, N. Ádám, J. Hurtuk, and E. Chovancova, "MLMD—A malware-detecting antivirus tool based on the XGBoost machine learning algorithm," Applied Sciences, vol. 12, no. 13, p. 6672, Jul. 2022, doi: 10.3390/app12136672. [Online]. Available: https://www.mdpi.com/2076-3417/12/13/6672
[18] P. Kumar, G. P. Gupta, and R. Tripathi, "Toward design of an intelligent cyber attack detection system using hybrid feature reduced approach for IoT networks," Arabian Journal for Science and Engineering, vol. 46, no. 4, pp. 3749–3778, 2021.
[19] A. Ekong, "Securing against zero-day attacks: A machine learning approach for classification and organizations' perception of its impact," Journal of Information Systems and Informatics, vol. 5, no. 3, pp. 1123–1140, 2023, doi: 10.51519/journalisi. v5i3.546.
[20] G. Naidu, T. Zuva, and E. M. Sibanda, "A review of evaluation metrics in machine learning algorithms," in Computer Science On-line Conference, Cham: Springer International Publishing, Apr. 2023, pp. 15–25.
[21] P. He, Y. Mao, C. Li, L. Cavallaro, T. Wang and S. Ji, "On the Security Risks of ML-based Malware Detection Systems: A Survey," ARXIV PREPRINT ARXIV:2505.10903, May 2025. [Online]. Available: https://arxiv.org/pdf/2505.10903. [Accessed: 19-Aug-2025].
[22] Kaggle, "Windows Malwares Dataset," [Online]. Available: https://www.kaggle.com/datasets/joebeachcapital/windows-malwares.