International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Malware Detection Via Classification

Guided by: Prof. Ayodhya Patil
Supriya Kshirsagar, Atharava Chaugule, Sarthak Kadam
Zeal Polytechnic, Pune, Maharashtra, India Academic Year 2026–2027 |

Abstract
Malware continues to threaten the confidentiality, integrity and availability of computer systems. This paper presents a machine-learning classification framework for identifying malicious activity from memory-analysis features in the CIC-MalMem-2022 dataset. Six supervised classifiers are considered: Random Forest, Decision Tree, Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) and XGBoost. The proposed workflow covers dataset inspection, preprocessing, feature-label separation, model training, validation, comparative evaluation and deployment through a Python Flask application. The dataset contains benign and malware memory records and includes malware categories such as ransomware, spyware and Trojan horse. A controlled train-test procedure and common classification measures accuracy, precision, recall, F1-score and confusion matrix are specified to make the comparison reproducible. Since experimental output from the project implementation was not supplied during preparation of this manuscript, no model accuracy or winning classifier is claimed. The paper instead defines the evaluation protocol and provides a reporting structure in which measured results can be inserted after execution. The intended contribution is an integrated and transparent workflow connecting memory-feature classification with a lightweight web interface for submitting feature records and displaying model predictions.

Keywords Malware detection, machine learning, classification, CIC-MalMem-2022, memory forensics, Random Forest, XGBoost, Flask.

I. INTRODUCTION

Malware is software designed to perform actions that are unauthorized or harmful, including data theft, disruption, surveillance and extortion. Detection is difficult because malicious programs can change their appearance, conceal execution and reuse legitimate system components. Signature-based tools remain useful for known threats, but a detector based only on previously observed signatures may fail to recognize unfamiliar or modified samples. Machine-learning classification offers another approach: it learns statistical relationships between recorded system characteristics and known labels, then applies those relationships to previously unseen records.

Memory analysis is relevant because running programs leave traces in volatile memory. Process information, loaded modules, handles, injected regions and other operating-system artefacts can provide evidence of suspicious activity. Dener, Ok and Orman studied malware detection using memory-analysis data and evaluated several learning algorithms on CIC-MalMem-2022 [1]. Later work has also explored classifiers for obfuscated malware using the same dataset [2]. These studies motivate further comparison, but their reported metrics cannot be transferred to a separate implementation: performance depends on preprocessing, feature selection, data splitting, parameter choices and evaluation design.

This project develops a classification-based malware detection workflow using six algorithms and a Flask interface. Its objective is to compare the models under the same data preparation and test conditions, identify practical trade-offs and expose predictions through a simple web application. The work is designed as a supervised classification task. In the binary setting, the target distinguishes benign records from malware records. If the project uses the dataset's more detailed category label, a separate multiclass experiment can distinguish benign, ransomware, spyware and Trojan records; that experiment must be reported separately from binary detection.

II. DATASET AND RELATED WORK

A. CIC-MalMem-2022 Dataset

CIC-MalMem-2022 is a memory-based malware dataset made available by the Canadian Institute for Cybersecurity. A published study describes 58,596 records, with 29,298 benign and 29,298 malicious samples, and reports memory-derived attributes associated with process lists, dynamic-link libraries, handles, loaded modules, services and related system artefacts [1]. The malicious records cover ransomware, spyware and Trojan families. The dataset is therefore suitable for studying whether patterns in memory features can separate benign activity from malicious activity.

The exact file version and column set used in this project should be recorded before training. Published descriptions sometimes refer to 55 numerical features plus label-related columns, while dataset distributions and exported CSV versions can differ in the number of visible columns. Accordingly, this paper does not assume a fixed feature count for the downloaded project file. The implementation should record the source URL, download date, filename, row count, column count, target column and class distribution in its experiment log.

B. Related Work

Dener et al. evaluated machine-learning and deep-learning approaches on CIC-MalMem-2022 and reported strong results for several classifiers, including Logistic Regression [1]. Cevallos-Salas et al. examined binary and multiclass classification for obfuscated privacy malware using the same dataset and discussed the use of traditional classifiers and a deep neural network [2]. These works establish the relevance of memory-derived features and provide context for the present comparison. The current project differs in its selected six-model comparison and in its integration of the trained classifier with a Python Flask interface. It does not claim to outperform the cited studies without a directly comparable experiment.

III. PROBLEM STATEMENT AND OBJECTIVES

The problem addressed is the automated assignment of a malware label to a record represented by memory-analysis features. Manual inspection of large feature tables is time-consuming, while fixed rules may not capture combinations of indicators. The project therefore investigates whether supervised classifiers can learn useful decision boundaries from labelled memory records and make predictions through a web interface.

A. Objectives

The objectives are: (1) prepare CIC-MalMem-2022 data for supervised learning; (2) train six classification algorithms using a consistent split and preprocessing pipeline; (3) evaluate each model using multiple metrics rather than accuracy alone; (4) compare error patterns, especially malware records incorrectly classified as benign; and (5) integrate a selected, saved model into a Flask application for prediction.

B. Scope

The study is limited by the coverage and representativeness of the dataset and should not be interpreted as a complete endpoint security product.

IV. PROPOSED METHODOLOGY

A. System Workflow

The proposed workflow consists of data acquisition, validation, preprocessing, model training, evaluation, model selection and web deployment. First, the dataset is loaded from its CSV file. The program checks duplicate rows, missing values, data types, target labels and unexpected values. Next, the target label is separated from the predictor columns. Only valid predictor features are passed to the learning pipeline; filename or other identifier columns should be excluded unless a documented experiment justifies their use, because identifiers can introduce spurious correlations.

The data is divided into training and testing subsets using a fixed random seed. Stratified splitting is recommended so that class proportions are retained in both subsets. All preprocessing operations that learn parameters from data—such as imputation, scaling and feature selection—must be fitted on the training data only and then applied to validation and test data. This prevents information from the test set leaking into training. A scikit-learn Pipeline can keep preprocessing and model estimation together [3].

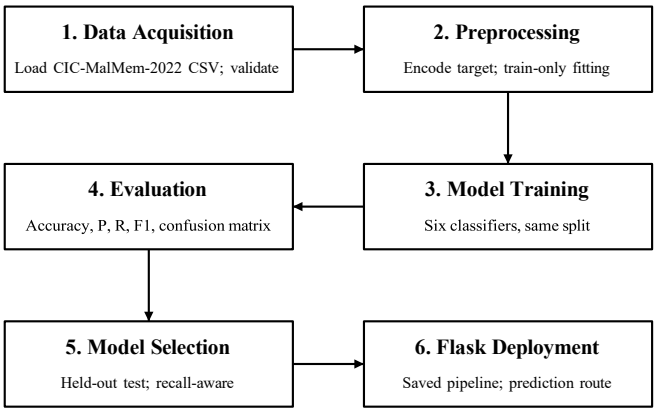


Fig. 1. Overall workflow of the proposed malware detection platform.

B. Preprocessing and Feature Handling

The target field is encoded consistently. Numeric predictors are checked for missing or infinite values and converted to appropriate numeric types. If categorical predictors are present, they require a suitable encoder fitted on the training data. Scaling is important for distance- and margin-based algorithms: StandardScaler is appropriate as a starting point for Logistic Regression, KNN and SVM. Tree-based methods such as Decision Tree, Random Forest and XGBoost generally do not require standardization, although the same cleaned feature set should be used for a fair comparison.

Any feature selection must be performed inside the training process rather than before the split. The project should avoid balancing or oversampling the full dataset before splitting. If imbalance is observed in the actual target distribution, class weights or resampling may be applied only to training folds. The published dataset is commonly described as balanced for the benign-versus-malware task [1], but the actual downloaded file must be checked.

C. Classification Algorithms

Random Forest constructs an ensemble of decision trees and combines their outputs. It can model nonlinear feature interactions and is generally less sensitive to individual tree variation. **Decision Tree** learns a sequence of feature-based splits and offers an interpretable set of decision rules, but a deep tree can overfit. **Logistic Regression** estimates class probabilities from a weighted combination of input features; it provides a useful linear baseline and can be regularized.

K-Nearest Neighbors assigns a class using the labels of nearby training observations. Its predictions depend on the distance measure, the value of k and feature scaling, and inference can become expensive for large training sets. **Support Vector Machine** seeks a separating boundary with a large margin; kernel functions can model nonlinear boundaries, while feature scaling and parameter selection are important. **XGBoost** is a gradient-boosted tree method that builds trees sequentially to reduce the errors of earlier learners. Its performance depends on parameters such as learning rate, tree depth, number of estimators and regularization.

D. Training and Model Selection

All six models should use the same training and test partition. Hyperparameters may be selected using cross-validation on the training subset only. A suitable baseline is stratified five-fold cross-validation, with a fixed seed and a documented search range. The final selected configuration is refitted on the full training subset and evaluated once on the held-out test subset. Report software versions, random seed, split ratio, hyperparameters and hardware environment so that the experiment can be repeated.

E. Flask-Based Deployment

After evaluation, the selected preprocessing-and-classifier pipeline is serialized as one model artefact. The Flask application loads this artefact at startup. A prediction route accepts feature values through a form or JSON request, validates required fields, checks numeric ranges and constructs the input in the exact feature order expected by the pipeline. The application then returns the predicted class and, when supported by the estimator, a probability or confidence score. Input validation errors should produce a clear response rather than a server exception.

The web interface is a demonstration layer and does not itself improve classifier quality. It should not execute uploaded files or accept arbitrary binaries. The application should use safe configuration, avoid exposing stack traces, limit request size and clearly label the result as an automated model prediction rather than a definitive forensic verdict.

V. EXPERIMENTAL DESIGN AND EVALUATION

The primary experiment is binary classification: benign versus malware. If a multiclass experiment is performed, it should be presented as a separate task with its own class-wise metrics. The recommended initial split is 80% training and 20% testing with stratification and a fixed random state; an alternative split may be used if the project code already defines one, but it must be reported accurately. The test set must remain untouched during model selection.

Performance should be measured using accuracy, precision, recall and F1-score. Accuracy measures the fraction of all predictions that are correct. Precision measures the fraction of predicted malware cases that are actually malware. Recall measures the fraction of actual malware cases detected by the model. F1-score is the harmonic mean of precision and recall. The confusion matrix should also be reported because false negatives—malware records labelled benign—are particularly important in a security application. ROC-AUC may be included when the model provides suitable probability scores or decision values. These metrics are supported by standard machine-learning evaluation libraries [3].

For the binary task, define malware as the positive class. Then TP denotes malware correctly detected, TN denotes benign correctly identified, FP denotes benign records incorrectly flagged as malware, and FN denotes malware records missed by the classifier. The formulas are: Accuracy = (TP + TN)/(TP + TN + FP + FN); Precision = TP/(TP + FP); Recall = TP/(TP + FN); and F1 = 2 × Precision × Recall/(Precision + Recall). If a denominator is zero, the metric implementation and zero-division policy should be stated.

A. Results Table to Complete After Running the Code

Table I is intentionally left without numerical values. Populate it with measurements generated by the project on the held-out test set. Do not insert values from another publication or from training accuracy.

TABLE I. PERFORMANCE COMPARISON OF THE SIX CLASSIFIERS

Classifier	Accuracy	Precision	Recall	F1-score
Random Forest	[insert]	[insert]	[insert]	[insert]
Decision Tree	[insert]	[insert]	[insert]	[insert]
Logistic Regression	[insert]	[insert]	[insert]	[insert]
KNN	[insert]	[insert]	[insert]	[insert]
SVM	[insert]	[insert]	[insert]	[insert]
XGBoost	[insert]	[insert]	[insert]	[insert]

VI. DISCUSSION

The best model should not be selected from accuracy alone. In a malware detector, a model with slightly lower accuracy but higher malware recall may be preferable if the cost of missed malware is high. Conversely, a very high false-positive rate can overwhelm analysts and reduce the usefulness of the application. The confusion matrix, class-wise precision and recall, inference time and model size should therefore be considered together.

The comparison is meaningful only when all classifiers use the same data partition and compatible preprocessing. In particular, KNN and SVM are sensitive to feature scale, while tree ensembles can exploit nonlinear interactions without standardization. Differences in preprocessing can otherwise make a model comparison misleading. Any result should be interpreted as performance on the selected dataset split, not as a guarantee of detection against every malware family or future threat.

The main limitations are dataset dependence, possible similarity between records derived from related samples or environments, and the difference between offline tabular classification and live endpoint detection. Random splitting can overestimate generalization if near-duplicate or related samples occur across both partitions. Where identifiers or acquisition groups are available, a group-aware split is a stronger additional test. The model also depends on the quality and consistency of memory feature extraction upstream.

VII. CONCLUSION AND FUTURE SCOPE

This paper has presented an original workflow for malware detection via classification using CIC-MalMem-2022, six supervised learning algorithms and a Python Flask interface. The methodology emphasizes controlled preprocessing, leakage prevention, consistent evaluation and practical deployment. No experimental winner or accuracy value is asserted because the project's measured outputs were not supplied. Once the implementation is executed, the results table and discussion should be completed using the held-out test metrics.

Future work can extend binary detection to classification of ransomware, spyware and Trojan categories; evaluate group-based or time-aware splits; investigate feature importance and explainability; test robustness against distribution shift; and measure prediction latency under realistic request loads. Deployment should be further assessed for input validation, logging, access control and secure model updates.

ACKNOWLEDGMENT

The authors acknowledge the Canadian Institute for Cybersecurity for making the CIC-MalMem-2022 dataset available for research and educational use.

REFERENCES

[1] M. Dener, G. Ok, and A. Orman, “Malware detection using memory analysis data in big data environment,” Applied Sciences, vol. 12, no. 17, Art. no. 8604, 2022, doi: 10.3390/app12178604.

[2] D. Cevallos-Salas, F. Grijalva, J. Estrada-Jiménez, D. S. Benítez, and R. O. Andrade, “Obfuscated privacy malware classifiers based on memory dumping analysis,” IEEE Access, vol. 12, pp. 17481–17498, 2024, doi: 10.1109/ACCESS.2024.3358840.

[3] Scikit-learn Developers, “Metrics and scoring: quantifying the quality of predictions,” scikit-learn documentation. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html. Accessed: Oct. 30, 2026.

[4] Canadian Institute for Cybersecurity, “CIC-MalMem-2022 dataset,” University of New Brunswick. [Online]. Available: https://www.unb.ca/cic/datasets/malmem-2022.html. Accessed: Oct. 30, 2026.