iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IX **Month of publication:** September 2026

DOI: https://doi.org/10.22214/ijraset.2026.84975

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IX Sep 2026- Available at www.ijraset.com

Explainable Malware Detection Framework Using Memory-Based Machine Learning, Windows Live Monitoring, and Containment

Cheekurumelli Rupa Sri[1], Dr.Kuda Nageswara Rao[2]
Department of Computer Science and Engineering, Andhra University, Visakhapatnam, Andhra Pradesh

Abstract: ***The advancement of malware in areas such as obfuscation and fileless is among the important factors affecting the efficiency of static analysis. The memory analysis technique has the potential to provide insights into how processes operate and detect attributes of possible malicious software; something which can only be achieved partially by static analysis. Many machine learning methods utilizing memory analysis characteristics have been tested in offline experiments, thus creating a gap between Windows endpoint online analysis and offline malware detection. Furthermore, lack of interpretability in many machine learning techniques makes the use of memory analysis features hard in malware detection. This research work offers a transparent algorithm for the detection of malware that consists of adaptability, process containment, process monitoring on Windows OS, interpretation using SHAP, and a memory-centric machine learning framework. Malware detection framework suggested by this study is trained using ObfuscatedMalMem2022 dataset, which includes 58,596 data instances along with 52 memory features, collected using Volatility software. In order to perform the classification, a stacked ensemble classifier is developed using Random Forest, XGBoost, and LightGBM as base classifiers, while Logistic Regression is used as the metaclassifier. The proposed ensemble achieves 98.17% accuracy, 98.43% precision, 97.92% recall, 98.17% F1-score, and 0.9822 AUC-ROC on the held-out test set. SHAP-based analysis is employed to interpret the model predictions and determine which memory-derived properties have the greatest influence, with process-handle and service-related attributes being the most important contributors. To implement the offline method for real-time endpoint monitoring, a Windows process monitoring module based on psutil gathers run-time process telemetry, leveraging feature adaptation to generate the feature representation necessary for the classifier. As some Volatility-based features cannot be monitored via user-space monitoring, the missing features are represented by benign baselines from training data. The generated risk score is then used to classify the process as being one of three types: safe, suspicious, and malware-risk. The Framework provides mechanisms for process suspension and executable isolation for high-risk processes. Thus, the viability of the integration of an explainable memory-based malware classification system with Windows live process monitoring and containment is demonstrated, yet the mismatch between training features based on memory and available telemetry is the most significant limitation to live operation.***

Keywords: Malware Detection, Memory Based Machine Learning, Ensemble Learning, Explainable AI, Live Monitoring, Containment.

I. INTRODUCTION

Cybersecurity environment is defined by continuous confrontation of threats with those who protect the system from them. Malware has evolved over time from simple executables to sophisticated forms that employ cutting-edge techniques like polymorphism, metamorphism, encryption, and even fileless execution [1], [2]. These methods have the ability to change the static features of malicious software, making detection based solely on predefined signatures and static properties increasingly challenging [1], [2].

Malware, because of the fact that it interacts with the operating system when it is running, creates valuable evidence in RAM. Consequently, memory-based analysis has emerged as an important approach for malware detection and classification [3]. Memory analysis methods can detect various artifacts of an executing program, which are usually hard to spot during static analysis but memory analysis methods can be used to detect them. The Volatility framework is commonly used to analyze memory dumps and identify fileless and memory-resident malware [3], [4].

Machine learning algorithms are being widely used for malware detection because of their ability to determine the features dependencies and distinguish between malicious and benign occurrences [5], [6].

©IJRASET: All Rights are Reserved

Interpretability is a problem when very complicated machine learning algorithms are used in cyber security. In case of malicious processes and samples, it is necessary to find out which input attributes are used by the model to arrive at that conclusion. Without the use of XAI, it becomes quite difficult to make sense of the predictions of sophisticated models [7]. Besides interpretability, another issue relates to the move from offline malware categorization to online monitoring and automated response. Previous research has been carried out on detecting and automatically terminating malicious processes in real time [8]. Malware behavior analysis has also been explored as a way to understand and identify malicious activities [9].

Applying a machine learning approach based on memory-related characteristics to a live Windows environment presents a distinct and technically challenging task. Memory features obtained from offline memory analysis datasets usually are obtained with the help of Volatility, while live process telemetry data is obtained in a completely different way. For example, live monitor can obtain runtime information such as process identifiers, CPU utilization, memory consumption, thread counts, and handles; However, this is not possible in the case of directly reproducing all memory-based features that can be accessed during offline analysis. This is because the training of any such model would be done using memory-based features, while at the same time the testing would use different features.

This paper presents an explainable malware detection framework that combines memory-based machine learning, SHAP explainability, Windows live process monitoring, feature adaptation, and automatic containment to address the above stated problems. The methodology will create a stacking ensemble model using the features derived from the memory using the Obfuscated-MalMem2022 dataset, using SHAP to identify the role played by the features in the predictions made by the model. Live telemetry from Windows process is gathered and analyzed through the process of feature adaptation to obtain the feature representation needed by the model. Features which cannot be directly obtained from the live data are provided by utilizing the training-data baseline that consists of the benign dataset. The prediction is used for risk categorization and process control purposes. The main contributions of this work are summarized as follows:

1) Ensemble-based malware detection: A memory-based malware detection framework is developed using the Obfuscated-MalMem2022 dataset. A Stacking Ensemble of Random Forest, XGBoost, and LightGBM with a Logistic Regression meta-learner is used to differentiate between benign and malware samples.

2) Explainable malware detection: SHAP is integrated to explain the model's decisions by identifying the memory-based features that have the greatest influence on malware classification, providing both global and individual prediction explanations.

3) Windows live monitoring and feature adaptation: A Windows live-monitoring component is developed to monitor running processes. A feature-adaptation mechanism maps the available runtime information to the 52-feature format required by the trained malware detection model.

4) Containment: Implementation of an interactive incident response pipeline designed to suspend and terminate malicious processes, isolate threat artifacts, and perform system registry verification to ensure complete and safe recovery.

II. RELATED WORK

A. Traditional and Behavioural Malware Detection

In initial strategies to detect malware, static analysis was used mostly, and it referred to the examination of the structure of the executable files without running them. However, more advanced obfuscation, packing and code loading techniques could reduce the efficiency of such detection strategies based only on static methods [1], [2]. As a result, malware behavior analysis has become an area of great academic interest for researchers studying how malware interacts and behaves during execution with the underlying OS. Yu et al. [9] give a survey of malware behavior description and analysis techniques covering different ways to represent and analyze malicious behavior. More recent work has studied system-call based analysis, including risk-based grouping of system-call sequences, for malware intrusion detection [10]. While behavioral analysis can produce useful runtime data, it is generally required to continuously monitor the execution activities and therefore contrasts with memory analysis that extracts artifacts from the volatile state of the system.

B. Memory-Based Malware Detection

Fileless and memory resident malware present additional problems for traditional detection based on disk because malicious activities take place mostly in volatile memory. This problem is solved by memory analysis that studies evidence from RAM images. Sihwail et al. [3] investigated memory analysis for malware detection and classification showing that memory-derived artifacts are useful for telling samples from benign ones. Vekariya et al. [4] examined memory forensics with the Volatility framework focusing on malware.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

These studies show that volatile-memory artifacts are sources of information for malware detection. Kara and Yılmaz [12] introduced a method that mixes static, dynamic and memory analysis, for fileless malware again highlighting memory analysis as a key part of malware detection. Nonetheless,memory-based approaches usually require memory images.one must pull out features that cannot be seen with standard live user-space monitoring.

C. Machine Learning and Ensemble-Based Malware Detection

Many researchers have studied machine learning for malware detection. This is because machine learning for malware detection can find patterns in different features instead of just looking at simple signatures [5], [6]. Almuqren et al. [5] looked into how machine learning algorithms can help with automated malware detection. At the same time Brown et al. [6] studied how machine learning can work with learning for automated malware detection.

Some researchers have also looked at learning for malware detection. This method works by joining classifiers together to make the results better. Imamverdiyev et al. [14] studied learning techniques to find hidden or obfuscated malware on Windows. Their work shows that ensemble learning techniques do work for finding obfuscated malware. Torres et al. [15] looked at malware detection by using feature engineering and behavior analysis. This study shows that picking the features is very important for machine learning for malware detection. All of these studies make me think that using methods, on the structured memory-based feature space I am studying here is a good idea.

D. Explainable AI for Malware Detection

The rise of using machine learning algorithms for identifying malware has led to the need for making understandable predictions. Galli et al. [7] investigated explainability in AI-based behavioral malware detection systems, highlighting the importance of XAI for understanding model decisions in security applications. Yu et al. [16] provide a systematic review of explainable AI for malware analysis and discuss different interpretability methods and their applications. Among these approaches, SHAP offers contributions at the feature level, which can be used for understanding the impact of each input feature on predictions made by the model. In terms of malware classification based on memory data, this helps in knowing what memory features play a vital role in making a classification decision. However, explainability only solves a portion of the issue; it does not solve the associated deployment difficulty.

E. Windows Live Process Monitoring

Whereas malware detection using memories is usually performed on collected images of the memory, endpoint monitoring aims at viewing the system behavior while running. Zeebaree [17] investigated process monitoring techniques at the operating-system level, establishing the relevance of process-level information for monitoring running systems. Some recent research has focused on live and behavioral approaches to detect fileless malware. Bandara et al. [18] investigated analysis for fileless malware detection and Pal et al. [19] investigated approaches for detecting malware. I see that these studies show how important runtime monitoring is for identifying threats, during execution and they also highlight that runtime monitoring can catch attacks in time. However, live monitoring systems generally observe a different set of telemetry from the memory-derived features available in offline forensic datasets. Consequently, directly applying an offline memory-based classifier to live process telemetry remains a non-trivial deployment problem.

F. Automated Malware Response and Remediation

Malware detection can be improved by using automated response systems that help contain a threat quickly after it is found. These mechanisms ensure that there is as little time as possible between identifying an issue and resolving it. The faster a threat is contained, the less damage it can do. Automated response mechanisms work well with malware detection to make security stronger. Rhode et al. [8] investigated real-time malware process detection together with automated process killing, demonstrating the integration of detection and process-level response.

Sun et al. [20] proposed a fast-and-slow architecture for real-time malware detection and mitigation and illustrated how automated response mechanisms can cut the time, between detection and response. These approaches demonstrate that automated response can extend malware detection systems beyond classification toward active threat containment. However, the response mechanisms in such systems do not necessarily address the feature-space mismatch that arises when a classifier trained on offline memory-derived features is connected to live Windows process telemetry.

G. Research Gap

Existing research has made progress in memory-based malware detection [3], [4]. Machine-learning-based classification has also improved, as shown by [5], [14]. Explainable malware detection is another area of advancement shown by [7], [16]. Live process monitoring has evolved, as noted in [17]–[19]. Finally automated malware response has seen progress, documented by [8], [20]. However, these research directions commonly address different stages of the malware detection and response pipeline. In particular, machine-learning models trained on rich memory-derived features extracted from offline memory images cannot necessarily consume the same feature representation from live Windows process monitoring. Conversely, live monitoring approaches may provide runtime telemetry without incorporating the richer memory-derived feature representations and explainability mechanisms available in offline ML-based analysis.

Therefore, an important unresolved deployment challenge is the adaptation of an offline memory-based malware classifier to live Windows telemetry when the training and deployment feature spaces do not fully match. This work tackles the challenge by using a memory-based stacking classifier, SHAPbased explainability, Windows live process monitoring, a feature-adaptation mechanism for related features and automated process containment all, inside a single unified framework.

III. PROPOSED METHODOLOGY

A. System Architecture

The proposed framework consists of two interconnected stages: an offline model-development stage and an online Windows monitoring and containment stage. The overall architecture is illustrated in Fig. 1.

During the offline stage, the Obfuscated-MalMem2022 dataset is preprocessed and scaled before being used to train a stacking ensemble consisting of Random Forest (RF), XGBoost (XGB), and LightGBM (LGBM) base learners with Logistic Regression as the meta-learner. SHAP-based analysis is incorporated to interpret the contribution of memory-derived features to model predictions.

During the online stage, running Windows processes are monitored using psutil. The collected runtime telemetry is processed by a Feature Adapter that constructs a 52-feature representation compatible with the trained classifier. Since several memory-derived features cannot be directly obtained through standard user-space process monitoring, benign baseline values are used for unavailable features. The adapted feature vector is subsequently scaled using the trained preprocessing pipeline and passed to the stacking classifier. The resulting malware probability is converted into a risk category and can trigger containment for high-risk processes.

B. Dataset Description

The machine-learning model is trained using the Obfuscated-MalMem2022 dataset. The dataset contains 58,596 samples, consisting of 29,298 benign and 29,298 malware samples. It has 52 numerical memory-derived features that reflect data from several memory-analysis fields.

The feature set includes process-related information such as pslist, handle-related information such as handles, injectedmemory characteristics represented by malfind, module and process-view information including ldrmodules and psxview, and Windows service-related information represented by svcscan.

The malware classifier is trained using these features, which make up the offline feature space. Because the feature representation obtained from memory analysis differs from the runtime telemetry available through psutil, a Feature Adapter is introduced to construct a model-compatible representation for live monitoring.

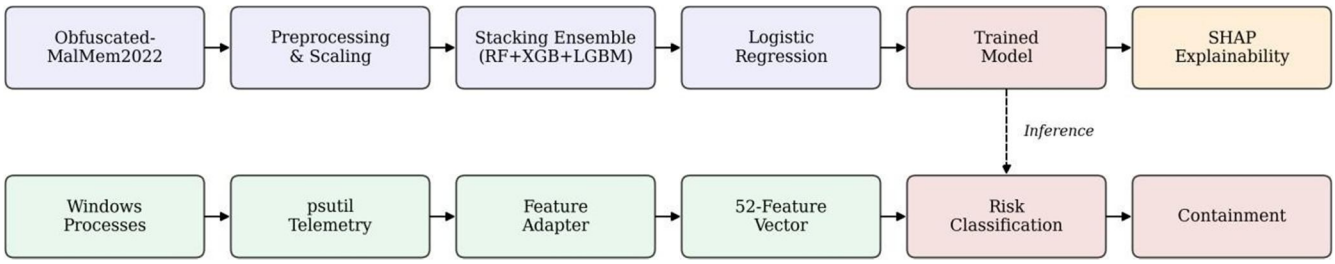


Fig. 1. Architecture of the proposed memory-based malware detection framework with SHAP explainability, Windows live monitoring, feature adaptation, and containment.

C. Data Preprocessing

Before the model is trained, the dataset is preprocessed to produce a consistent numerical representation.

1) Feature Selection: The target label is separated from the input attributes, and the numerical feature columns required by the classifier are selected. Constant-valued features are excluded from the model input.
2) Data Splitting: A stratified 70/15/15 split is used to separate the dataset into subgroups for testing, validation, and training. To maintain the distribution of classes among the three subgroups, stratification is employed.
3) Missing-Value Handling: Missing numerical values are handled using column-wise median imputation. The preprocessing parameters obtained during model development are retained for subsequent transformations.
4) Label Encoding: For supervised machine-learning classification, the malware and benign class labels are encoded into numerical representations.
5) Robust Feature Scaling: RobustScaler is used to lessen the impact of outliers and account for the various numerical ranges of memory-derived characteristics.

The value of the scaled feature is computed as:

$$X_{scaled} = \frac{X - Q_2}{Q_3 - Q_1} \tag{1}$$

where Q_1, Q_2, and Q_3 represent the 25th, 50th (median), and 75th percentiles. The fitted scaler is subsequently reused when transforming validation, test, and live-process feature vectors.

D. Ensemble Machine-Learning Model

A stacking method is used for aggregating predictions from different machine-learning models. Three tree-based base learners—Random Forest, XGBoost, and LightGBM—are included in the first level.

1) Random Forest (RF): Random Forest uses randomized feature selection and sampling to integrate several decision trees. By using an ensemble approach, the variance related to individual decision trees can be decreased.
2) XGBoost (XGB): Gradient boosting is used by XGBoost to build a series of decision trees, each of which improves on the mistakes of the models before it. Model complexity is managed and generalization is enhanced by regularization procedures.
3) LightGBM (LGBM): A gradient-boosting technique called LightGBM was created for effective learning using structured numerical input. Model training is made computationally efficient by its histogram-based learning process.

Meta-Learner: The predictions generated by the three base learners are combined using a Logistic Regression metalearner. A 5-fold cross-validation strategy is used to generate out-of-fold (OOF) probability predictions from the base learners. These OOF predictions form the meta-feature matrix used to train the Logistic Regression classifier.

The meta-learner therefore learns how to combine the complementary predictions of the three base models to produce the final classification.

When enabled, the base learners' hyperparameters are optimized using Optuna. The optimization evaluates candidate configurations using 3-fold cross-validation with ROC-AUC as the optimization objective. The optimized configurations are subsequently used to construct the stacking ensemble.

To classify the specific malware family, the system focuses on Ransomware, Spyware, and Trojans, as these correspond directly to the three obfuscated threat categories defined in the Obfuscated-MalMem2022 training dataset. Once the primary machine learning model flags a process as malware, the system extracts its process name, executable path, and command-line arguments. By scanning this metadata for known suspicious keywords, the framework instantly calculates and assigns the threat type without requiring additional machine learning computation.

E. Explainable AI Module

The approach integrates a SHAP (SHapley Additive exPlanations) to enhance the machine-learning predictions' interpretability.

Each feature is given a contribution value by SHAP according to how much it contributes to a model prediction. For a given prediction, the SHAP value of feature j can be expressed as:

$$\phi_j(x) = \sum_{S \subseteq N \setminus \{j\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \Big(f(S \cup \{j\}) - f(S) \Big) \tag{2}$$

where N represents the complete set of features and S represents a subset of features excluding feature j. The relevance of memory-derived features to the tree-based models' predictions is examined using the SHAP module. The methodology produces local explanations to analyze the impact of specific characteristics for certain predictions and global feature-importance information to find influential features throughout the dataset. Thus, in addition to the numerical classification findings, SHAP visualizations offer an extra interpretability layer.

F. Windows Live Process Monitoring and Feature Adaptation

The online component provides live monitoring of running Windows processes using the psutil library. A background monitoring process periodically collects available runtime process information and passes it to the Feature Adapter.

The principal challenge is that the offline classifier is trained using memory-derived features, whereas standard user-space process monitoring cannot directly reproduce all of the memory-analysis artifacts. Therefore, the Feature Adapter constructs a 52-feature representation compatible with the trained classifier.

The adaptation process consists of the following mechanisms.

1) Runtime Process Mapping: Runtime process data available at process level are gathered using psutil. The total number of threads that can be obtained from the process under consideration is mapped to handles.nthread in the input data model.

2) Proxy Feature Mapping: Some runtime measurements are used as proxies for related memory-derived features. For example, the virtual memory size obtained from process.memory_info().vms is converted to megabytes and used as a proxy representation for malfind.commitCharge. These mappings do not suggest that the runtime analysis is the same as the original memory analysis feature; instead, they represent the mapping of an approximation to enable the live telemetry to be analyzed using the classifier.

3) System-Level Feature Representation: The adapter also constructs system-level feature values required by the model, including process-, thread-, handler-, service-, driver-, and module-related attributes. Where the required information is not dynamically available through the live user-space monitoring mechanism, the configured baseline representation is used.

4) Benign Baseline Values: Features that cannot be directly observed or reasonably approximated using the available live telemetry are initialized using benign baseline values derived from the benign training representation. This encompasses memory analysis capabilities that rely on data not accessible via typical process inspection tools, such as specific process view, module, callback, and memory analysis properties.

5) Model-Compatible Feature Vector: After the available runtime values, proxy values, system representations, and benign baselines are combined, the adapter constructs the complete 52-feature vector expected by the trained classifier.

The feature vector is arranged according to the feature ordering used during model training and is then passed through the fitted preprocessing and scaling pipeline. Thus, the Feature Adapter does not reconstruct the complete memory state of a running Windows system. Instead, it uses baseline values, proxy mappings, and accessible data to produce a model-compatible representation of live process telemetry.

G. Threat Classification and Containment

The adapted live-process feature vectors are processed using the trained preprocessing pipeline and passed to the stacking ensemble. The classifier produces a malware probability: P(y = 1|x), where y = 1 represents the malware class. The resulting probability is converted into three operational risk categories:

$$\text{Risk} = \begin{cases} \text{Malware} & P \geq 0.60 \\ \text{Suspicious} & 0.35 \leq P < 0.60 \\ \text{Safe} & P < 0.35 \end{cases} \quad (3)$$

For the end-user live monitoring dashboard, this probability P is multiplied by 100 and presented as a Risk Score percentage

Processes classified as malware undergo the following containment and remediation steps:

1) Threat Identification and Data Protection: The system checks the process command-line arguments to identify the malicious script and distinguish it from legitimate programs such as python.exe. It also checks active file handles to identify affected user files and creates a backup before taking further action.

2) Process Containment and Termination: The malicious process is first suspended using psutil to stop it from performing further actions. It is then terminated to remove the active threat and release its memory.

3) Artifact Isolation: After termination, the related malicious executable or script is moved to a secure quarantine directory. This prevents further execution while preserving the file for later analysis.
4) Persistence Verification: The system checks Windows Startup Run Keys and Task Scheduler to determine whether the malware has created any mechanisms to restart automatically after a system reboot.
5) Recovery and Audit Logging: The system keeps a recovery copy of the backed-up files. It also records the containment actions, process ID (PID), executable path, threat classification, and timestamps in system logs and generates a Markdown incident report.

IV. EXPERIMENTAL SETUP

A. *Hardware and Software Environment*

Python 3.11 was used on a Windows endpoint environment for each experiment. The framework is built on top of Scikitlearn, xgboost, lightgbm, shap, and optuna. SSE is used for monitoring in the Flask backend. Parallelism was limited (n_jobs=1) to avoid memory fragmentation for ensemble inference on Windows.

B. *Training Configuration*

Optuna used a 20% training data subsample and was set up for 15 trials per base model with a 120-second timeout. The combined Train + Validation set (n = 49,806) produced 5-fold OOF probabilities that were used to train the Stacking MetaLearner (Logistic Regression).

C. *Evaluation Metrics*

The models were assessed using the held-out test set (n = 8,790). Accuracy, Precision, Recall, F1-Score, False Positive Rate (FPR), False Negative Rate (FNR), and Area Under the Receiver Operating Characteristic Curve (AUC-ROC) are examples of standard binary classification metrics that were computed.

V. RESULTS AND DISCUSSION

A. *Model Performance*

To determine the effectiveness of the proposed malware detection technique, three tree-based machine learning algorithms—namely Random Forest (RF), XGBoost (XGB), and LightGBM (LGBM)—were evaluated using the memory-derived features presented in Section III. The selected models served as base learners for the proposed Stacking Ensemble model. The obtained results are presented in Table I. All three individual models achieved an accuracy above 98%, demonstrating that the memory-derived feature representation contains discriminative information for distinguishing benign and malware samples.

Table I. Performance comparison of base models and proposed stacking ensemble

Model	Accuracy (%)	Precision (%)	Recall	F1-Score (%)	ROCAUC
Random Forest	98.13	98.21	98.05	98.13	0.9817
XGBoost	98.18	98.29	98.07	98.18	0.9817
LightGBM	98.12	98.18	98.06	98.12	0.9818
Stacking Ensemble	98.17	98.43	97.92	98.17	0.9822

The precision and recall values marked for the individual base models are approximate values derived from the available evaluation results. The primary reported evaluation measures for the base-model comparison were F1score and ROC-AUC.
Among the individual models, XGBoost achieved the highest reported accuracy and F1-score, both at 98.18%. LightGBM achieved the highest ROC-AUC among the individual models at 0.9818. The results show that all three base learners exhibit good and fairly consistent performance in classifying the features extracted from memory.

B. *Ensemble Model Performance*

The proposed Stacking Ensemble combines the predictions of Random Forest, XGBoost, and LightGBM through a Logistic Regression meta-learner. The final ensemble was evaluated on a completely unseen test dataset containing 8,790 samples.
The Stacking Ensemble achieved an accuracy of 98.17%, precision of 98.43%, recall of 97.92%, F1-score of 98.17%, and ROC-AUC of 0.9822.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

The resultant confusion matrix included 4,300 true negatives (TN), 4,329 true positives (TP), 69 false positives (FP), and 92 false negatives (FN).

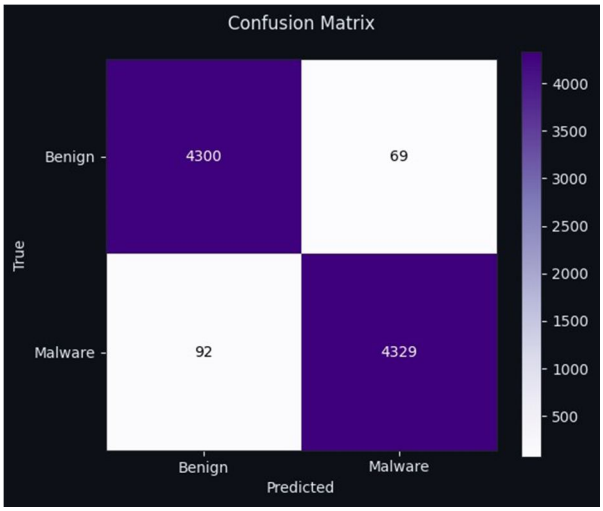


Fig. 2. Confusion matrix of the proposed Stacking Ensemble.

Based on these values, the false-positive rate (FPR) was 1.58%, while the false-negative rate (FNR) was 2.08%. Only a small percentage of benign samples were incorrectly identified as malware, according to the comparatively low false-positive rate.

The ROC curve of the Stacking Ensemble is presented in Fig. 3.

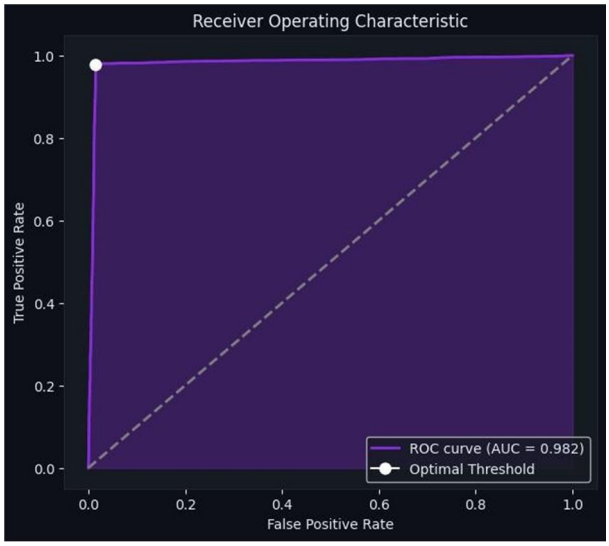


Fig. 3. ROC curve of the proposed Stacking Ensemble.

The resulting ROC-AUC of 0.9822 indicates strong discriminatory capability between benign and malware samples on the evaluated test dataset.

It should be noted that the Stacking Ensemble does not achieve the highest numerical accuracy among all reported individual-model values, since XGBoost achieved 98.18%. Therefore, the ensemble is not described as universally superior to every base learner. Rather, the results show that the ensemble performs similarly to the best performing single classifier while utilizing the predictions of multiple classifiers, and also attains the highest ROC-AUC and precision values yet reported.

TABLE II Stacking Ensemble Test Set Performance(N=8,790)

Metric	Value
Accuracy	98.17%
Precision	98.43%
Recall (Sensitivity)	97.92%
F1-Score	98.17%
AUC-ROC	0.9822
False Positive Rate (FPR)	1.58%
False Negative Rate (FNR)	2.08%

C. SHAP Explainability Results

To increase the explainability of the malware classification model, SHAP (SHapley Additive exPlanations) approach was used to examine the effect of memory-related features on the model's decisions.

The global SHAP analysis identified the following five features as the most influential:

Among the analyzed features, svcscan.nservices produced the largest mean absolute SHAP value of 1.6411, indicating that it had the strongest overall influence on the model's predictions.

Table III. Top 5 Features By Mean Absolute Shap Value

Rank	Feature (Volatility Plugin)	Mean |φ|
1	svcscan.nservices	1.6411
2	svcscan.process_services	0.5256
3	handles.avg_handles_per_proc	0.4239
4	svcscan.shared_process_services	0.3972
5	callbacks.ncallbacks	0.0731

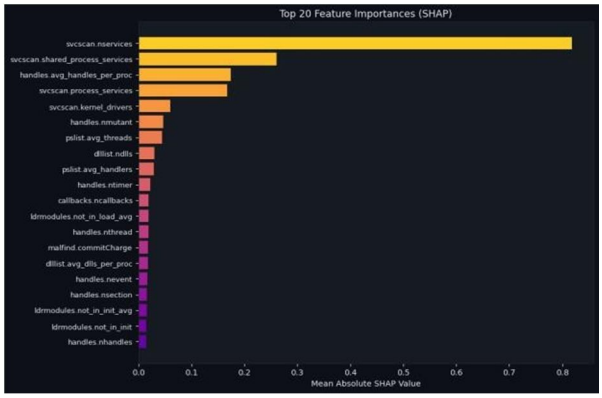


Fig. 4. Global SHAP feature importance of the proposed malware detection model.

The other highly influential features were svcscan.process_services, handles.avg_handles_per_proc, and svcscan.shared_process_services.

The prevalence of service-related attributes suggests that the model's learnt classification patterns heavily rely on service-related memory information.

In a similar vein, the comparatively high contribution of handles.avg_handles_per_proc suggests that process-handle attributes also significantly influence the model's predictions.

Local SHAP explanations for specific predictions offer further details about the characteristics that support or contradict a certain classification.

By demonstrating how individual feature contributions affect a particular prediction, the local SHAP explanation enhances the global feature-importance analysis. This makes the interpretation of individual model decisions more transparent.

The SHAP results should be interpreted as explanations of the model's learned decision process. A high SHAP contribution indicates that a feature influences the prediction; it does not, by itself, prove the presence of a particular malware technique.

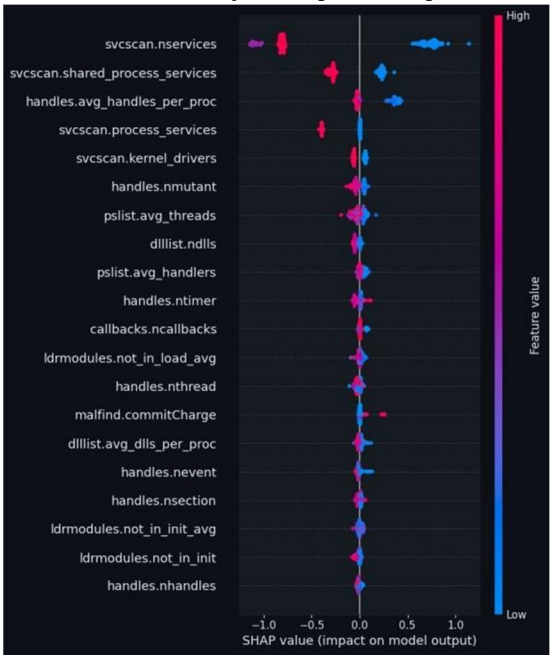


Fig. 5. SHAP waterfall explanation for a representative prediction.

D. Live Monitoring and Containment Framework

The Windows-based live monitoring and remediation systems were created to serve as a prototype for integrating the trained malware classifier into the running Windows operating system environment. The two modules were not included as a performance analysis of the quantitative model outlined in Sections IV-A and IV-B.

The prototype continuously monitors running Windows processes using the psutil library. Runtime process information is collected and passed to the Feature Adapter implemented in the _extract_process_features function of the live-monitoring component.

The trained machine-learning pipeline expects a 52-feature vector, which the Feature Adapter builds in accordance with that feature schema as the offline model was trained using memory-derived features obtained through Volatility, several of the original features cannot be directly obtained from standard psutil process telemetry. The system uses easily obtained runtime data for observable attributes, as well as mappings for unobservable features.

This vector of features is then fed to the preprocessing pipeline and the Stacking Ensemble model in order to obtain the malware probability. The prototype categorizes processes into three risk levels as described in Equation (3).

For processes classified as malware, the prototype executes containment workflow. The system preemptively backs up affected user data, suspends and terminates the high-risk process to halt execution, and securely isolates the malicious executable into a quarantine environment before verifying system integrity.

Therefore, the live-monitoring section showcases how the entire process of trained classifier, feature adjustment, risk assessment, and containment can be practically integrated in the Windows platform.

However, no independent accuracy, precision, recall, F1score, or ROC-AUC is reported for the live-monitoring prototype. Such metrics would require a dedicated real-time evaluation dataset containing representative live-process telemetry and verified ground-truth labels. Accordingly, the live-monitoring component is presented as a proof-of-concept Framework rather than a quantitatively validated real-time detection system.

E. Discussion

From the experimental results, it is clear that the tree-based model used performed reliably well on the memory extracted features, since all the three separate models used achieved accuracies of more than 98%. XGBoost obtained the highest individual accuracy of 98.18%, while the proposed Stacking Ensemble achieved 98.17% accuracy, 98.43% precision,

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

97.92% recall, 98.17% F1-score, and the highest ROC-AUC of 0.9822. Although the ensemble did not achieve the highest numerical accuracy, it maintained performance comparable to the strongest individual model while combining the predictions of Random Forest, XGBoost, and LightGBM. The confusion matrix further showed a low false-positive rate of 1.58%, indicating that relatively few benign samples were incorrectly classified as malware. The SHAP analysis identified svcscan.nservices, svcscan.process_services, handles.avg_handles_per_proc, svcscan.shared_process_services, and callbacks.ncallbacks as the most influential features, demonstrating that service-, handle-, and callback-related memory information contributed substantially to the learned classification decisions. These explanations improve the transparency of the detection model by showing which features influence its predictions. While the SHAP values should be considered explanations for the model behavior instead of definitive proof of malware techniques used, the live-monitoring for Windows part is an example of how a trained model can be implemented to perform a real-life Windows processes monitoring task through runtime telemetry and Feature Adapter. However, since the live-monitoring part has been created as a Framework. Generally, the results obtained have shown the efficiency of the suggested machine learning approach based on memory.

VI. LIMITATIONS

The offline malware classification system had a number of drawbacks despite its good performance:

1) User-Space Telemetry Gap: The Windows live-monitoring component relies primarily on user-space telemetry obtained through psutil. Several of the features found in the memory-based training dataset, including ldrmodules and psxview, cannot be accessed by the direct use of the traditional process monitoring API. Hence, the Feature Adapter utilizes the available measurements as well as the baseline features that cannot be accessed directly. As a result, the telemetry provided during live monitoring differs from the feature space used during offline training.

2) Limited Live-Environment Validation: The Windows live-monitoring and containment components are currently implemented and demonstrated as a prototype rather than as a fully validated real-time detection system. The reported 98.17% accuracy and 0.9822 ROC-AUC apply to the offline model evaluation and should not be interpreted as live detection performance. Comprehensive evaluation of the live system using a controlled set of verified malicious and benign processes is required to measure its real-time detection effectiveness, false-positive behavior, response latency, and containment reliability.

3) Dependence on Feature Adaptation: Because the live monitor cannot directly reproduce the complete set of memory forensic features used during training, the effectiveness of live predictions may differ from the offline evaluation. The use of baseline values for unavailable features provides compatibility with the trained model but does not represent direct observation of the corresponding runtime memory structures. A possible future implementation, with better Windows telemetry or even access to memory level feature extraction, may help bridge this gap between feature space.

4) Heuristic-Based Threat Typing: To categorize the specific malware family (e.g., Ransomware or Trojans), the prototype currently relies on a lightweight, keyword-based heuristic engine rather than a dedicated multiclass machine learning model. While this ensures rapid real-time performance, it is limited by predefined signatures. Therefore, novel malware families that do not match the configured metadata keywords may only receive a generic malware classification.

The above are some of the limitations facing the suggested framework today. This gives clear indication on the direction that future research should take.

VII. CONCLUSION AND FUTURE WORK

This research presented an explainable malware detection framework that integrates containment, Windows live process monitoring, SHAP-based interpretability, and memory-based machine learning into a single architecture. On the examined unseen test dataset, the suggested Stacking Ensemble achieved 98.17% accuracy, 98.43% precision, 97.92% recall, 98.17% F1-score, and 0.9822 ROC-AUC by integrating Random Forest, XGBoost, and LightGBM via a Logistic Regression meta-learner. A Windows live-monitoring Framework was created to showcase the integration of runtime process telemetry, feature adaption, risk assessment, and containment in addition to the offline detection and explainability components.

The main goal of future work will be to close the feature-space gap between live Windows telemetry and the memory-based training environment. To lessen reliance on baseline representations for features that are currently unavailable through psutil, more thorough Windows telemetry and deeper memory-level feature acquisition will be explored. In order to quantify real-time detection accuracy, false-positive behavior, response latency, and containment dependability, the live-monitoring and containment components will also need to be systematically evaluated in an isolated environment using controlled, confirmed malware and benign-process datasets.

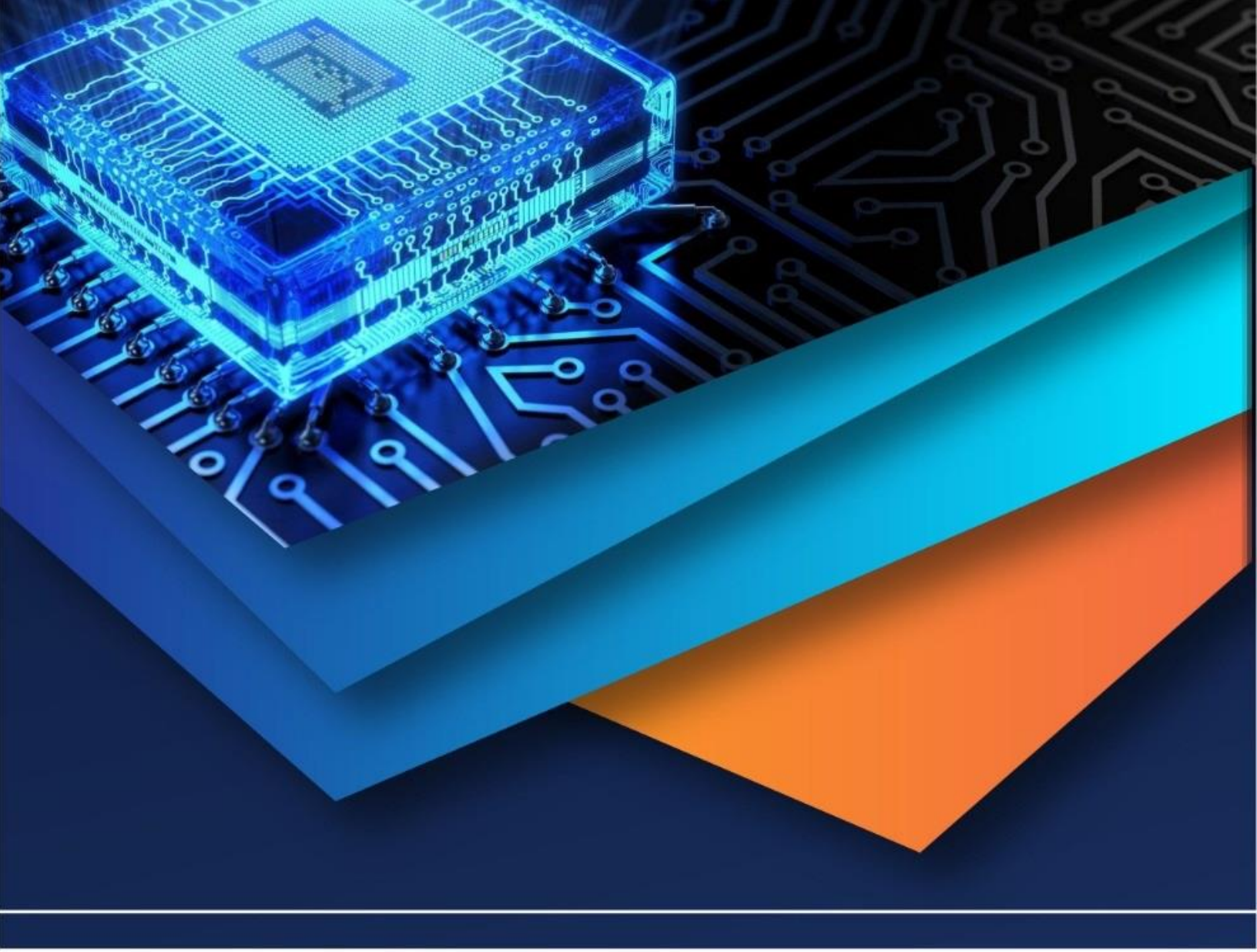
©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

To assess how well the suggested approach generalizes outside of the current experimental environment, other research can look into larger datasets and different malware types. Additionally, future iterations of this framework could replace the heuristic threat-typing engine with a dedicated multiclass machine learning model, allowing the system to dynamically categorize entirely novel malware families without relying on predefined execution keywords.

REFERENCES

[1] J. Ferdous, R. Islam, A. Mahboubi, and M. Z. Islam, “A Review of
[2] State-of-the-Art Malware Attack Trends and Defense Mechanisms,” IEEE Access, vol. 11, pp. 121118–121141, 2023.
[3] S. Chandran, S. R. Syam, S. Sankaran, T. Pandey, and K. Achuthan, “From Static to AI-Driven Detection: A Comprehensive Review of Obfuscated Malware Techniques,” IEEE Access, vol. 13, 2025.
[4] R. Sihwail, K. Omar, and K. A. Z. Ariffin, “An Effective Memory Analysis for Malware Detection and Classification,” Computers, Materials & Continua, vol. 67, no. 2, pp. 2301–2320, 2021.
[5] D. Vekariya, M. Patidar, M. Amin, and C. Padhiyar, “Memory Forensics
[6] Using the Volatility Framework: A Structured Approach for Detecting
[7] Fileless Malware,” in 2025 IEEE 5th International Conference on ICTBIG, 2025.
[8] A. Almuqren, M. Frikha, and A. Albuali, “Automated Malware Detection Based on a Machine Learning Algorithm,” in 2023 IEEE 10th Int. Conf. on ComNet, 2023.
[9] A. Brown, M. Gupta, and M. Abdelsalam, “Automated Machine
[10] Learning for Deep Learning Based Malware Detection,” Computers & Security, vol. 137, 103582, 2024.
[11] A. Galli, V. La Gatta, V. Moscato, M. Postiglione, and G. Sperlì,
[12] “Explainability in AI-based behavioral malware detection systems,” Computers & Security, vol. 141, 103842, 2024.
[13] M. Rhode, P. Burnap, and A. Wedgbury, “Real-Time Malware Process Detection and Automated Process Killing,” Security and Communication Networks, vol. 2021, 8933681, 2021.
[14] B. Yu, Y. Fang, Q. Yang, Y. Tang, and L. Liu, “A Survey of Malware Behavior Description and Analysis,” FITEE, vol. 19, no. 5, pp. 583–603, 2018.
[15] T. Vyšniunas, D. Čeponis, N. Goranin, and A. Čenys, “Risk-Based System-Call Sequence Grouping Method for Malware Intrusion
[16] Detection,” Electronics, vol. 13, no. 1, 206, 2024.
[17] S. Chidambaram and S. M. P., “MemLOL: Memory-Based LOLBins Dataset for Fileless Malware Detection,” in 2025 13th ISDFS, 2025.
[18] I. Kara and K. Yılmaz, “HAP-Analyzer: A Hybrid Static, Dynamic, and Memory Analysis Approach for Fileless Malware,” IEEE Access, vol. 14, 2026.
[19] B. Ç. Sarı and İ. F. Kılınçer, “Hybrid Deep Learning Model for Memory-Based Malware Detection,” in 9th Int. IDAP Symp., 2025.
[20] Y. Imamverdiyev, E. Baghirov, and J. C. Ikechukwu, “Detecting
[21] Obfuscated Malware Infections on Windows Using Ensemble Learning Techniques,” Informatics and Automation, vol. 24, no. 1, pp. 99–124, 2025.
[22] M. Torres, R. Álvarez, and M. Cazorla, “A Malware Detection
[23] Approach Based on Feature Engineering and Behavior Analysis,” IEEE Access, vol. 11, pp. 105355–105367, 2023.
[24] B. Yu et al., “Explainable AI for Malware Analysis: A Systematic
[25] Review of Benchmark Datasets, Traffic-Oriented Detection, and Interpretability Methods,” PeerJ Computer Science, vol. 12, e3902, 2026.
[26] S. R. M. Zeebaree, “A New Approach for Process Monitoring,”
[27] Polytechnic Journal, vol. 1, no. 1, 2011.
[28] H. Bandara et al., “Stealth Eye: Behavioral Analysis for Fileless Malware Detection,” in 13th ISDFS, 2025.
[29] B. Pal, G. H. Reddy, S. Saurav, and A. R. Vasudevan, “Detection of Fileless Malware,” in 2025 5th ICSC, 2025.
[30] R. Sun et al., “Learning Fast and Slow: PROPEDEUTICA for Real-time Malware Detection,” arXiv preprint arXiv:1712.01145, 2021.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)