SCIENTIFIC CULTURE
www.sci-cult.com

SCIENTIFIC CULTURE, Vol. 12, No. 4, (2026), pp. 7910-7925
Open Access. Online & Print

DOI: 10.5281/zenodo.12426790

STACKING ENSEMBLE MODEL WITH TUNED-UMAP-ENHANCED FEATURE SPACE: MALWARE DETECTION BASED API CALL SEQUENCES

S Sasikala, G Saranya*

Department of Computer Science Engineering,
Amrita School of Computing, Amrita Vishwa Vidyapeetham,
Chennai, Tamil Nadu, India
Email: sasi.umasasi@gmail.com, g_saranya@ch.amrita.edu

Received: 07/12/2025
Accepted: 02/04/2026

Corresponding Author: G Saranya
(g_saranya@ch.amrita.edu)

ABSTRACT

The rapid growth in Internet connections has led to a significant increase in cyberattacks, which can have severe consequences in real-time applications. Malware is one type of malicious program that causes various forms of cyberattacks. The conventional detection methods have difficulties in the detection of malware, such as polymorphic code and API call spoofing, because they can hardly keep pace with the ongoing development and evolving tactics. To address these limitations, this study proposes a behavioural malware detection framework, termed UMAP-EnS-ZT, which integrates manifold learning, ensemble detection, and Zero Trust enforcement within a unified architecture. The framework analyses dynamic API call sequences and applies Uniform Manifold Approximation and Projection (UMAP) to reduce high-dimensional behavioural features while preserving structural relationships among samples. The transformed representations are processed through a stacked ensemble model comprising Extra Trees(ET), Gaussian Naïve Bayes(GNB), K-Nearest Neighbours(KN), and XGBoost(XGB) as base learners, with HistGradientBoosting(HGB) serving as the meta-learner. This layered strategy improves generalization and reduces model variance across heterogeneous malware families. Beyond classification, the proposed system incorporates a Zero Trust mechanism, and a dynamic trust score is computed from behavioural outputs and predefined risk thresholds to enable continuous verification and policy-based execution control. This ensures that processes are not implicitly trusted after initial detection, thereby strengthening runtime containment. Experimental evaluation on the MalBehavD-V1 dataset demonstrates a detection accuracy of 96.26%, outperforming traditional detection approaches and single-model baselines. The results indicate that integrating behavioural ensemble learning with measurable Zero Trust enforcement provides a robust and adaptive solution for malware detection in dynamic threat environments.

KEYWORDS: Ensemble Staking Model, Malware Detection, API Call Sequences, Zero Trust, Threat Classification, Cyber-Attack.

1. INTRODUCTION

The significant increase in the development of cybersecurity assists in building creative techniques to identify and stop undesirable activity [1, 2]. These malevolent acts take advantage of weaknesses in people and organisations, such as phishing, identity theft, malware dissemination, and other similar practices [3]. This malware risk to computer networks and systems security has gradually become more common in recent years.

At the end of 2021, the privately owned IT privacy research group AV-TEST claimed more than 1.2 billion instances of malware had been recorded, with an estimated 350,000 new malware samples being found per day [4]. An advanced security measure with a detection method is needed for implementing the constant modification in malware enclosures, which is underscored by an alarming growth rate. Because fraudsters are using new strategies and tactics to avoid detection by conventional techniques, malware attacks have also become more sophisticated and complex. According to a Symantec report, supply chain attacks rose by 93% in 2021, ransomware attacks became profitable and determined, and targeted attacks climbed by 42%. These figures highlight the need for creative, astute, and automated methods to recognize and stop malware's increasing danger.

Any package that is purposefully created to damage a computer system and jeopardise user security is referred to as malicious software, or malware. If a software program or code executes malicious actions and surreptitiously acts against the needs of the computer user, it is deemed malware. It exploits a variety of platforms, including servers, PCs, smartphones, and webcams, in an effort to obtain illegal access, steal personal information, and interfere with the system's regular operation. For several years, the malware has been known for its detrimental activities along with attacks. Dynamic analysis of malware across various platforms is complex and modifies the task frequently [5].

The refined omnipresent growth, along with evasive malware [6], suggests the generation of high flexibility and resilience in measuring cybersecurity. In general, the conventional antivirus technique uses signature-based detection that has certain limitations, which are made worse by the divergence of various antivirus vendors [7]. The conventional signature-based technique and static analysis method have been determined to be inefficient against threats, with a decrease in detection rate below 60% in zero-day attacks [8]. This has possessed an essential interest in a dynamic method that analyse runtime behavior specifically through monitoring API calls. However, the existing solution of dynamic analysis consists of three crucial challenges discussed below

1. Limited detection scope: Most of the security method concentrated on either statistical feature such as missing temporal dependencies, or sequential patterns, which ignore the structural relationships, leading to incomplete analysis [9].
2. Scalability constraints: The scalability constraint remains, with 83% of the available model evaluated using a static method on datasets with fewer than 100,000 samples. The severe limitation is relevance in the real world. [10].
3. Explainability Gaps: This became an essential issue as commercial solutions frequently act as a black box, gathering efficient incident responses and hindering analyst trust [11].

Based on earlier studies, the limitation advance investigation into how strongly the antivirus software detecting along with eliminating malware infestations. Malware analysis from the behavioural viewpoint demonstrates potential in supporting cybersecurity actions [12]. This method involves cautious monitoring and investigation of the operations, together with communications presented by log files or programs, in identifying all potential questions or detrimental behaviour. If in the case of a malware signature, it is modified or unknown, the method permits the security technique for recognising and terminating the unwanted actions. One of the critical elements in dynamic analysis is malware's API calls analysis because it corresponds to behavioral patterns along with the operating system [13]. The main factor that enables communication among applications, along with OS, is the Windows API, which contains several operations inside the specific software libraries. To endorse that OS are legitimate in identifying the potentially malicious behaviour, which is critical for investigating the API call patterns as well as sequences [14]. However, there are several API calls utilized as a program for completing the tasks [15, 16]. Therefore, the research improves the capability in preventing novel security threats along with transforming the way in which malicious software gets recognised. Thus, the proposed EnS model performs better in analyzing malware behaviour that can be determined to be polymorphic, metamorphic, and zero- Trust. Moreover, the proposed model for API call analysis has established malware to dimension reduction, together with efficient maintenance of large datasets, and identifies intricate

linkages and trends with Zero trust for continues verification and least privilege access.

DR method minimizes the dataset's features during preservation as much as possible, with critical data. It includes converting high-dimensional data into a low-dimensional space that preserves the significant characteristics of the original data. Unlike Principal Component Analysis (PCA), which relies on a linear structure of attributes, UMAP is a non-linear approach that can better handle complex interactions among data properties. The UMAP approach helps create predictive models to improve MD findings and discover important factors influencing the arrangement of API calls. This study examines the effectiveness of the EnS model and uses a range of stacking techniques to increase the accuracy of the MD results with Zero trust. The effectiveness of the model is demonstrated by a precision-recall curve and visualisations of accuracy and loss during model training. A Confusion Matrix (CM) simultaneously demonstrates highly true positive and true negative rates, demonstrating precise and novel data classification, finally ZTA practices the rule of never trust and always Analysis. It continuously authenticates identity, behaviour, and customizing device posture of all users or systems before it is granted entry or access is sustained.

The structure of the paper is as follows: Section 2 discuss about review of dynamic malware analysis based on various implementations of API call sequences, detection of ML and DL models, and limitations. Section 3 describes the proposed methodology, which includes an EnS classifier, data preprocessing, API call sequence, and FS of the UMAP technique. Section 4 presents the parameters used for each algorithm and accuracy with other evaluation metrics for the proposed model as a comparative study and Zero Trust Mechanism. Section 4 discusses briefly the conclusion and future direction.

2. LITERATURE REVIEW

Research on MD and classification with Zero trust constitutes a significant area of investigation in the field of cybersecurity. Studies on malware classification using dynamic datasets have garnered increasing interest in recent years. This section reviews important studies in the literature associated with MD and classification.

The implementation and examination of malevolent samples in a measured setting is known as dynamic analysis. It is necessary to use fundamental system API calls to implement malicious behaviours [17,18]. System APIs are typically used to detect sandboxes, observe GUI operations, change the Windows registry, initiate network communication, and gain the necessary system rights. By keeping focus on these delicate procedures, dynamic analysis techniques are generally thought to be more resistant to influence [19]. Sequences of API calls are thought to be an illustrative method for comprehending the behavioural traits of malware. The malicious code is run in a controlled setting, like the Virmon or Cuckoo sandboxes, in order to acquire API call sequences [20]. Dynamic behaviours are constantly detected and ultimately traced during the execution process. Malware usually needs to start several processes in order to carry out particular actions. A C4.5 decision tree based on the sequence of API calls was used to build a supervised learning approach that distinguishes malicious files from benign ones [21]. [22] Employed a Random Forest (RF) technique for categorisation after extracting API calls together with additional behaviors, including file operations, registry alterations, and network usage. [23] Mined characteristics from network flow patterns to create a supervised malware classification based on RF and KNN. Although effective on the test set, these methods rely significantly on feature selection and require practitioners' knowledge to choose and improve appropriate features. It becomes increasingly necessary to strengthen feature engineering as harmful malware evolves and its resistance abilities increase.

Almushabb et al. [24] have identified the malicious tasks carried out by the malware, which are deduced through its API call sequences. Security, privacy, and system accessibility are the three aspects actively threatened by vigorous cyberattacks that can ensue from a failure in detecting these malware instances. By possessing this hybrid method, this research seeks to advance the MD-based efficiency and precision with respect to API calls, although it addresses the conventional weight initialization method's disadvantages. Employing a balanced dataset, the experimental results demonstrate an accuracy as 83% and a loss of 0.44, which outperforms the standard models with respect to minor loss. [25] Look at a trivial, order-invariant way to find and stop malware intimidations by looking at API calls lacking concern about their order. The dataset is over 550GB when uncompressed. This model utilises ML algorithms like random forests and behavioural analysis to look at anomalies and patterns in API call sequences. This model is not only effective but also efficient. The suggested model

achieves a remarkable F1 score of more than 85% while operating on any platform with minimal performance overhead. [26] A new MD model called Mal ASSF is proposed that combines the semantic and sequence aspects of the API calls. The representation of the DR of the API function is obtained using the API2Vec embedding approach. Balts is used to extract the behavioral aspects of successive segments in order to capture them. Operation and resource type are retrieved in order to take advantage of the implied semantic data of the API purposes. Mal-ASSF performs 3% to 5% better than current solutions in terms of detection accuracy, according to the study using a dataset of malware families.

DL can automatically recognise malicious behaviours and detect sequence patterns [27]. As a result, it is a useful instrument for addressing the problem of malware classification [28, 29, 30]. Karat et al. [31] showed effectiveness in detecting malware that is specifically designed to evade detection methods. With a focus on its importance in addressing the continually varying field of cybersecurity challenges, this study attempts to evaluate the effectiveness of the Convolutional Neural Network-Long Short-Term Memory (CNN-LSTM) model. The method successfully combines the memory-efficient properties of LSTM models with the fast-processing speed of CNNs. All of these mechanisms work together to enable the process to efficiently handle datasets of a significant size. Scalability is especially important when real-time analysis is involved, as the ability to make prompt recommendations is essential to stopping malware from spreading and minimizing potential harm. [32, 33] focuses on the most sophisticated type of malware, known as metamorphic malware. Due to the fact that metamorphic malware cannot be detected by anti-virus software using conventional signature-based techniques, it is challenging to categorise this sort of malware appropriately. This study's classification technique is LSTM, a popular technique for categorising consecutive data. The classifier produced results that show accuracy of up to 95% with a respectable F1-score of 0.83. [34] Describes increasing malware classification accuracy and adaptability to provide a Deep Learning (DL) architecture reinforced with a genetic algorithm. To continuously improve the DL model and guarantee resilience against changing malware threats, our method integrates transformation processes and fitness score assessments inside genetic algorithms. The experimental results demonstrated notable gains in classification accuracy and loss reduction, suggesting that our method has the potential to improve MD in lively portable executable settings with concept drift behavior. [35, 36] uncovers hidden components in the hybrid characteristics of Android applications by using the FA method. A DNN with two hidden layers is fed the features that were extracted by FA after being enlarged using the DL technique. In the suggested approach, the DL strategy enhances the DNN architecture's learning capacity, which has a high computing capacity for validating the method from the Kronodroid dataset. Application of benign and malware compounds to the Kronodroid dataset that has been generated particularly for investigating and observing cross-device detection issues. According to the Androzoo dataset, the accuracy for the proposed method is 98.40%. [36] Have proposed the novel hybrid DL method that integrates GANs together with Gated Recurrent Units (GRUs) for improving MD through Windows portable executable file API call sequences. Based on several datasets, this research work progressed to compare the GRU-GAN method in contrast to alternative methods, namely Bidirectional LSTM (BiLSTM) as well as Bidirectional Gated Recurrent Unit (BiGRU). In the case of a complex dataset, the hybrid model has performed better than others, as determined through the evaluated results with an accuracy 98.9%. Because of less memory usage together with rapid training and testing time consumption, it advances better than current models with respect to resource utilisation. Ma et al. have proposed a novel approach to MD that accomplishes the use of identified direct relationships in API sequences. This method integrates structural data, node properties, and directional communication for modelling each API call as a concentrated graph [37]. Functions can be estimated through the Directed Graph Convolutional Network (DGCN). This research work illustrates the first-order and second-order graph convolutional networks (FSGCN) for effective capturing of these features. After that, the FSGCN's directed graph embeddings are converted into grayscale pictures and identified by a CNN [38] employed to identify malware, utilising the advantages of API calls. Researchers discovered that the MLP algorithm produced the greatest detection performance, with an accuracy of 91%. Though the overall classification accuracy was limited because this study only examined the conventional characteristics of API calls and did not investigate the linkages between API invocations. A GCN was used by [39] to create a method for detecting and classifying Android malware. Their method, GDroid, identified 98.99% of malware for Android with less than 1% false positives by projecting Android APIs and applications into a big, assorted graph and approaching the issue as a node cataloguing problem.

Limitation in existing works: DL models have been utilized extensively in recent decades for recognizing dangerous behaviours and extract sequence properties. Recent research, however, has shown that both static and dynamic features can be tricked by packing and black-box attacking techniques [40 - 42]. This problem is caused by two factors. First of all, merely converting APIs to numerical values misses a function's intrinsic semantic properties. Network interaction the file system, or other elements may be involved in an API function that does a query or modification action. These characteristics cannot be captured by simple numerical values. Second, the sequential qualities are not effectively reflected by the models that are currently in use. Large datasets are beyond the capabilities of these models. When these models are given a dataset with a high amount of API categories, large feature sets, and lengthy sequences, their performance generally declines. As a result, the API call sequences are typically altered to disregard important API data, which lowers the effectiveness of detection.

In summary, the functionality calls of API sequences used in the present work on dynamic feature analysis are insufficient to adequately depict the behavioural patterns of harmful groups. The current classification approach's performance falls short of what is needed to correctly categorize different families of harmful programs. As a result, the UMAP-EnS model used in this study shows a remarkable 96% accuracy rate in identifying hazardous activities.

3. PROPOSED MODEL

One ML algorithm might not be able to collect data and predict behaviour with any degree of accuracy. To increase accuracy, we employ four distinct classifiers as a base learner, feeding the meta learner with their output. In this section, the workflow of the suggested methodology will be described. Four classifiers are included in our suggested stacking model: ET, GNB, KN, XGB classifier at level 0 (base layer), and HGB at level 1 (meta layer). Figure 1 details our proposed model's entire workflow. The details of the classifiers utilized in stacking are as follows:

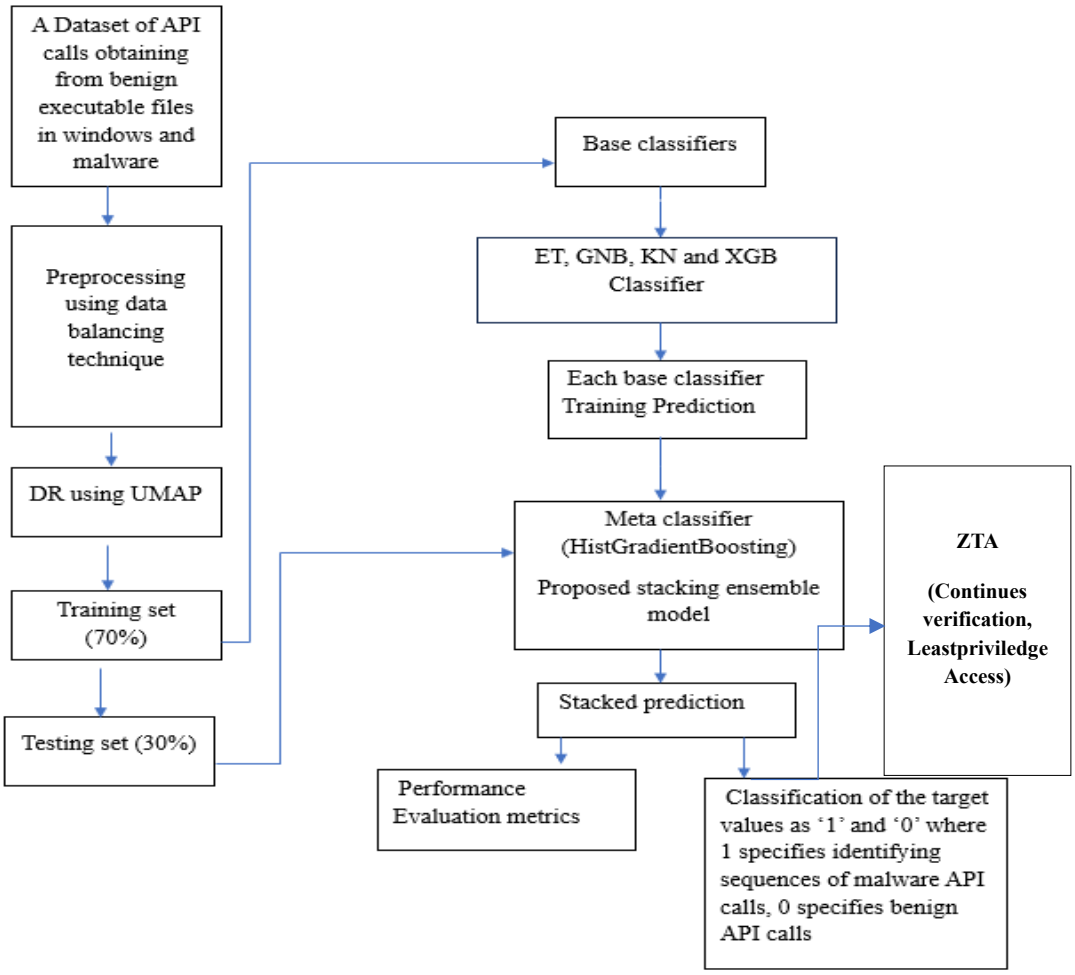

Figure 1. A suggested EnS model for malware API call classification and identification.

The dynamic datasets, which comprise permissions and API requests, are being evaluated on EnS models and ML. Highly successful MD results are obtained via analyses carried out with dynamic datasets. Dynamic analysis makes it possible to identify harmful activity by tracking the actions taken by the software and how it affects the system while it is running. Dynamic datasets are essential in this setting to comprehend malware's real-time activities and resource usage. Zero Trust enforcing is based on dynamic analysis. The API monitoring and permission use behavioural outputs are converted into quantitative risk indicators that are added to an ever-changing trust score. This score controls execution control by policy making, so that no tacit faith is placed on processes, in any of the operating phases. Rather than making binary access decisions, the system uses conditional verification using observed dynamics and resource usage patterns which is facilitated by dynamic behavioral modeling combined with computation of measurable trust to detect malware, which goes beyond classification in adaptive containment. This assimilation supports the execution of the responsibility and integrates the behavioral knowledge with enforceable principles of Zero Trust, enhancing the vulnerability to emerging and unforeseen threats.

3.1 API Call Sequence

Application Programming Interface (API) call sequences are fine-grained behavioural execution traces that can be used to describe the interaction of software and the operating system at run time. In contrast to static features, API sequences encode temporal dependencies and execution contextual patterns to give a behavioural fingerprint that is partially resilient to obfuscation, packing, and polymorphic transformations. As such, sequence-based model behaviour has since emerged as a mandate paradigm in the identification of previously unknown malware variants.

A significant percentage of Windows malware uses Dynamic-Linked Libraries (DLLs) to become maliciously executed payloads in legitimate processes by remote injection processes. OpenProcess is used to obtain a handle to the target process, VirtualAllocEx is used to allocate memory in the remote address space, WriteProcessMemory is used to write the payload path, and CreateRemoteThread is used to start the execution. The sequential execution of these APIs constitutes a typical execution pattern, which can be derived from dynamic traces and utilised as discriminative behavioural characteristics of malware classification.

This was done in this work, where API call sequences were obtained using dynamic execution traces of the malware and benign samples as a behavioural representation dataset. The auxiliary dataset with heterogeneous malicious and benign API execution sequences was introduced to make the model more It is robust against concept drift and unknown malware families and was trained on the aberrant behavioural patterns instead of a predefined signature. The obtained sequence embedding were then used to train the designed detection system in detecting novel and dynamic malware examples [43-45]. The transition to Zero Trust is not the change in policy only, it needs to be supported operationally. Each request of access has to be repeatedly authenticated depending on identity, device conditions, and situational behaviour. The advantage of this method is that it improves malware protection through strict authentication, tracking system usage, and restricting the performance of processes that are not authorized by the user in the system environment [46].

3.2 Dataset Description and Data Pre processing

The MalBehavD-V1 dataset was used in the given work, being a publicly available benchmark dataset of dynamic API calls traces of Windows executable files [47]. The dataset has 1,950 samples in total, including 1,650 malicious executable and 300 benign software samples. This distribution is representative of the asymmetry that is provided by real-world malware detection. The sequences of API calls of each executable file were downloaded using the calls part of the Cuckoo Sandbox execution logs. The top 100 non-repetitive API calls of parent process were chosen to represent the behaviour of the program in a fixed-length representation. Good software samples were obtained using the trusted software repositories like CNET and Virus total was used to obtain Malware software samples to capture the various malicious behaviours. In order to allow numerical processing, the API call names were converted to integer values by building a dictionary of API functions with 308 unique functions. An API function was given a different index, and the machine learning and deep learning models could directly accept such numbers as input. A pre-processing process was also conducted to check the quality and reliability of the dataset. Samples were removed in order to eliminate data leakage as well as similarity bias in samples. The distribution of classes was discussed to ensure that there was imbalance

between the malicious and benign samples. As the size of the malware was much more than that of the benign, so experiments were performed on the original imbalanced dataset and a balanced version of the dataset. Random underdamping was done to the malware group in order to build the balanced dataset. The model performance of the proposed model was reported in both datasets to determine how robust the model is in different circumstances of varying class distribution.

3.3 DR by UMAP

Uniform Manifold Approximation and Projection (UMAP) is a nonlinear dimensionality reduction model that is aimed at maintaining the intrinsic structure of a high-dimensional data set in a low-dimensional embedding space. UMAP is based on the theory of manifold learning and encodes the connections between the individual data points by building a weighted graph on the proximity of the neighbors. This paper used UMAP to derive the dimensionality of high dimensional representations of API call sequence before ensemble learning. The fact that API call sequences can be encoded as integer-valued vectors leads to sparse and high-dimensional feature space, and can affect model performance and generalization. UMAP achieves the objective of projecting these representations into a lower-dimensional latent space to allow more compact and informative feature representations whilst maintaining discriminative behavioural patterns. UMAP builds a fuzzy weighted k-nearest neighbour graph where the weight of the edges is the similarity between the data points. An embedding into a lower dimensional representation is then optimised to hold such relationships as close as possible. UMAP also captures nonlinear relationships in behavioural data as compared to its linear counterparts like Principal Component Analysis (PCA). Unlike t-SNE, UMAP offers a higher level of scalability and is more resilient to global data structure. The input to the proposed stacking structure of the ensemble was the low-dimensional feature representation achieved through UMAP. Such change makes the computational complexity less and enhances the stability of the classifier by alleviating noise and redundancy found in the original feature space. Ablation experiments also prove that UMAP plays a significant role in the overall performance of the suggested UMAP-EnS framework.

Model steps for the UMAP technique

The UMAP technique was used to minimize the dimensionality of high-dimensional API call feature representations. The processing steps can be summarized as follows.

Step 1: Data Preparation: The extracted API call sequences were represented as a feature matrix, where each sample corresponds to a row and each feature corresponds to a column. Prior to dimensionality reduction, the feature vectors were normalized to ensure comparable feature scales.

Step 2: Construction of Local Neighborhood Graph: The k-nearest neighbors of each sample were found in the original high dimension space. It is used to identify local relations between samples and to model the manifold structure of the data.

Step 3: Graph Representation: A weighted graph was created where nodes are the samples and the edge between the nodes are relationships of neighborhood. Similarity in pairs between samples was used to determine edge weights, obtaining a topological representation of data manifold on a fuzzy representation.

Step 4: Low-Dimensional Embedding Optimization: UMAP is an optimization of a low-dimensional mapping by reducing the distance between the high and low-dimensional fuzzy simplicial sets. This method maintains neighborhood relationships as well as world data structure.

Step 5: Low-Dimensional Feature Extraction: The optimized embedding produces compact low-dimensional feature vectors that retain discriminative structural information. These representations were used as input features for the subsequent ensemble learning model.

3.4 Proposed EnS classifier

Stacking is a method of ensemble learning in which a learner combines several heterogeneous base classifiers via a meta-classifier that learns how to best combine the output prediction levels of the base classifier. In contrast to the classical ensemble approaches, like bagging and boosting, stacking takes advantage of a hierarchical learning structure in which low-level models provide a prediction, which can be used as an input feature in a higher-level model. Under stacking, original feature vectors are presented to several base classifiers also known as level-0 learners. All the base classifiers generate prediction results, which are combined to generate a new feature representation. These prediction outputs are then used in training a level-1 meta-classifier which generates the final classification decision. This hierarchical structure allows the model to utilize complementary decision patterns learnt by the various classifiers and enhances a better generalization performance than the individual models.

The "caretStack" function is used in conjunction with a Generalized Linear Model (GLM) to build the stacked ensemble model, or "stack.model." The framework makes use of both the "Accuracy" metric and the combined prediction results from the previous stage. This model under "stackControl" is trained using the same tuning settings as the base-level models. Applying the "stack.model" to the stack predictions method is the next step. This function generates probabilities and makes use of "as.data.frame," which yields a frame with columns contrasting the actual findings from the testing set with the predicted anticipated results using the provided data from "stack.model."

The dataset is used as the input to build the stacking ensemble model. The dataset is

D = $\{ x_i , y_i \}_{i=1}^{m}$, where x_i signifies the feature vectors and y_i signifies their classification.

The model's initial phase comprises four base-level classifiers, h_t. The ET Classifier is h_1, GNB is h_2, XGB is $h_3$and KN is h_4. Using level 0, each base-level classifier $h_t$is trained. Level 0 refers to the input data from the training set that is used to generate predictions by level 0 classifiers. Creating a dataset that contains X_i in equation (1) comes next once the classification models have been trained.

$X_i = \{ h_1 (x_i), h_2 (x_i), h_3(x_i), h_4 (x_i) \ldots\ldots\ldots, h_T(x_i) \}$ (1)

The meta-classifier's new input in this phase is the predictions made in the previous stage. Applying and training a GLM to learn the meta-classifier is the last stage. HistGradientBoosting is the new classifier, h_M, which is based on the newly created dataset. Using equation (2), the final output is shown, with H representing the stacked ensemble model.

$H(x)=h_M(h_1(x), h_2(x), h_3(x), h_4(x) \ldots \ldots, h_T (x))$ (2)

The last step is to estimate the metrics based on proposed EnS model for classification and detection of malware API calls using a confusion matrix. In order to resample the features, we define the target values as '1' and '0' where 1 specifies identifying sequences of malware API calls, 0 specifies benign API calls.

Steps: Proposed UMAP-EGKH stacked ensemble model for classification

Determine the elements' pairwise distances from one another.

Build the neighbourhood graph.

To represent the high-dimensional data, create a fuzzy simplicial set.

Y is a low-dimensional space representation of F that is initially random.

while not converged:

update Y through the reduction of F and Y's cross entropy.

return Y

For each base learner B_i in B do

Training of Base Models for Predictions and Categorization:

If B_i is ExtraTreesClassifier then

1. The entire feature set, which is likewise chosen at random for every tree
2. Using extra trees involves choosing a feature's splitting value at random.
3. Rather than splitting the data using entropy or Gini to determine a locally optimal value obtain forecasts
4. A split value is chosen at random by the algorithm.
5. Get prediction P_{1k}

Else if B_i is GaussianNB

6. To use the algorithm to categorize every new data point determines each class's maximum posterior probability value.
7. Assigns that class to the data point.
8. Calculate the data point.
9. Get prediction P_{2k}

Else if B_i is KNeighborsClassifier

1. Choosing the optimal value of K
2. Scheming detachment
3. Discovery Nearest Neighbors
4. Voting for Classification or Enchanting Regular for Regression
5. Get prediction P_{3k}

Else if B_i is XGB

- $y_i$is the final predicted value for the ith data point
- K is the number of trees in the ensemble
- $f_k (x_i)$signifies the prediction of the Kth tree for the ith data point.

End if

End for

Concatenate prediction $P_k = \{ P_{1k} , P_{2k} , \ldots\ldots, P_{4k} \}$ using equation (2)

Meta model training:

6. As input features, use concatenated predictions P_k.
7. Train meta learner M (HistGradientBoosting) on P_k:
8. Get the prediction for the meta model M_k

end for return Final Predictions M_k

3.5 Zero Trust–Driven Decision Layer

In the proposed framework, the stacking ensemble is embedded within a Zero Trust–driven security architecture rather than operating as an isolated

detection component. Following the Zero Trust principle, no executable is implicitly trusted, and each execution request undergoes behavioural validation before being considered for deployment. The posterior probability generated by the ensemble model is treated as a behavioural risk estimate and converted into a continuous trust score. This trust score is supplied to a policy evaluation module that determines execution privileges, including normal execution, enforced sandbox isolation, or termination of the binary. Unlike static allow–deny policies, the framework continuously assesses behavioural risk based on API call patterns and dynamically updates execution decisions.

Decision-making and enforcement are decoupled through the implementation of a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP). The PDP evaluates the ensemble prediction outputs together with contextual attributes such as process lineage and historical behavioural patterns. The PEP is realized through the sandbox orchestration layer, which enforces the policy decisions by regulating executable deployment, sandbox isolation profiles, and execution termination. This design enables continuous verification and least-privilege execution for previously unseen binaries.

Algorithm: Zero Trust Enforcement
Input: Executable E at the Host System
Output: Allow / Isolate / Block
Step 1: Extract API sequence A from E ={ x_1, x_2, x_3…… x_n}
Step 2: Reduce A using UMAP → Z
Step 3: Predict P_mal = Ens (Z)
Step 4: Compute TrustScore (E) = 1 - P_mal
Step 5: if TrustScore ≥ θ_1 then Allow
Step 6: else if θ_2 ≤ TrustScore < θ_1 then Isolate
Step 7: else Block

3.6 Zero Trust–Driven Trust Modeling and Policy Decision

Instead of treating malware detection as a static classification task, the proposed framework integrates the ensemble output into a Zero Trust decision model. The predicted malware probability P_{mal} (E) for an executable E is transformed into a continuous trust score:

$$\text{Trust (E)} = 1 - P_{mal}(E) \quad (3)$$

To support continuous verification, trust is updated dynamically over time using a trust persistence factor α:

$$\text{Trust}_t(E) = \alpha \cdot \text{Trust}_{t-1}(E) + (1-\alpha)(1-P_{mal}(E)) \quad (4)$$

Mathematical Definition:

$$\textbf{Decision (E)} = \begin{cases} \boldsymbol{Allow}, & \boldsymbol{Trustscore(E) \geq \theta_1} \\ \boldsymbol{Isolate}, & \boldsymbol{\theta_2 \leq Trustscore < \theta_1} \\ \boldsymbol{Block}, & \boldsymbol{Trustscore(E) < \theta_2} \end{cases}$$

Where,

Table.1 shows the Trust (E) is the continuous trust score, θ_1 and θ_2 are security policy thresholds. This formulation enables continuous verification and risk-adaptive execution control in accordance with Zero Trust principles. Where α € [0, 1] is a persistence coefficient controlling trust inertia. In our experiments, α was set to X based on validation analysis.

Table 1: Decision Enforcement.

Decision Enforcement Action	
Allow	Export file to production allow-list
Isolate	Run in restricted VM profile (no network / limited privileges)
Block	Stop VM execution and flag sample

The enforcement layer is implemented through the Cuckoo Sandbox orchestration engine, which acts as a Policy Enforcement Point. Executable are never executed directly on the host system; instead, they are first evaluated in a sandbox environment. The stacking classifier serves as the Policy Decision Point by producing a behavioural risk score that is converted into a Zero Trust confidence metric. Based on predefined security thresholds, the sandbox controller determines whether the executable is permitted, isolated, or blocked. This design ensures continuous verification before deployment and adheres to the principle of least privilege.

4. RESULTS AND DISCUSSION

Malicious acts were replicated using a bespoke script. Sysmon logs are generated by the current script and then examined using established beforehand correlation criteria. The logs are given a degree of severity value based on the criticality of the discovered activity after a match is found. Furthermore, all API calls made throughout the simulation are captured and converted to integers, allowing for the training of an EnS model. Each of the 175 columns in the dataset represents a successive API call. A Python script is utilized to run database queries and export the results to a CSV file in order to perform a more thorough study of the data. Using a Python script for XML parsing, the Rohitab API Monitor program records API calls, extracts requests, and processes them. In a list of 308 unique API calls, the calls are given integer values. These values are

then fed into a UMAP DR method. After that, the supplied data is analysed and transformed into a format that works with the proposed EnS model. The pattern of the model can be learned through machine and generate an accurate prediction about features and intensity level for the identified activity using this data to train the model. The train data consist of labels that represent whether a particular outcome is expected coexist with a specific class. The main objective is to train the learning model for accurately identifying the unknown data location by contradictory to the historical data. However, the certain instance has disclosed in specific instance, the single learning model can able to produce the best results or the minor errors. Therefore, this experimental research utilizes EnS method that include in generating several hypothesis from the train dataset as well as integrating them for precise identification of sample location. Thus, the method improves the model's efficiency significantly through integration of various models that can improve the output accuracy. This method has yield a resilient and stable that shown high significant than individual models. For creating proposed EnS model, this experiment basically train each ML classifier into stacking model. The classifier involved in the earlier section involves ET, KN, XGB, HGB and GNB classifier that perform as a Meta-Learner classifier. A comprehensive learning process depends on the distinct structure, hyper parameters, and capabilities of each classifier. These classifiers serve as the foundation for the EnS approach following training. By employing the Stacking technique, this tactic enhances the EnS performance. Table 1 shows the parameters used for each algorithm in this section.

Table 2: Parameters of BL and ML.

Algorithm	Parameters
ET	n_est is 100 crit is gini max_dep is 2 rand_sta is 42
GNB	Priors = None var_smooth= 1e-9
KN	n_neighbors param =23, np.arange (2,30,1)
XGB Classifier	max_depth=3, subsam =1,scale_pos_weight>0
HGBClassifier	Pen = l2 Solver = lbfgs max_iter = 100
Stacking of Ensemble classifier	estim = ET, GNB, KN classifier, XGB classifier, final_estimator = HGB classifier

4.1 Performance Metrics

This study used staking techniques for malware dataset classification to ascertain whether an API request sequence was malicious or benign. A dataset of 1950 entries that has been split 70:30 across training and test sets is used for experimental testing. Our proposed models are evaluated using the following performance criteria: sensitivity, specificity, recall, accuracy, and precision. The following formulas (5), (6), (7), and (8) use the ideas of True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) as the basis for these measures:

The fraction of accurately predicted samples compared to all samples is known as accuracy.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (5)$$

Precision is defined as the percentage of expected positive samples that actually.

$$\text{Precision} = \frac{TP}{TP+FP} \quad (6)$$

The percentage of accurately predicted positive results compared to all expected positive samples is known as the recall.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (7)$$

Calculating the weighted average of Precision and Recall yields the F1-score.

$$\text{F1-score} = 2 * \frac{Precision*Recall}{Precision+Recall} \quad (8)$$

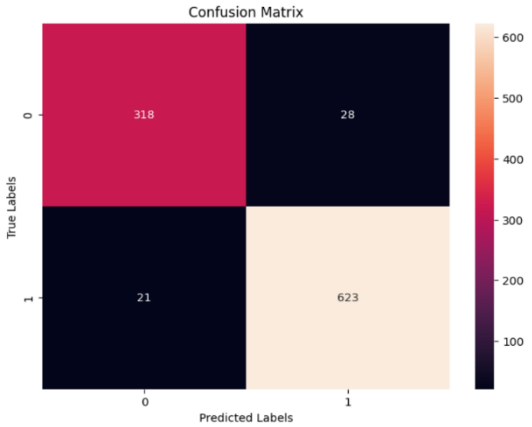

Figure 2 CM for proposed EnS of model

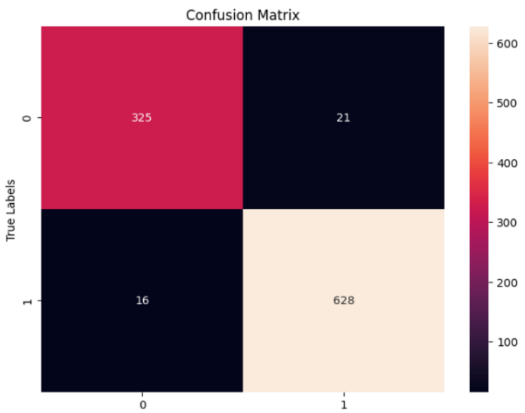

Figure 3 CM for ET Classifier

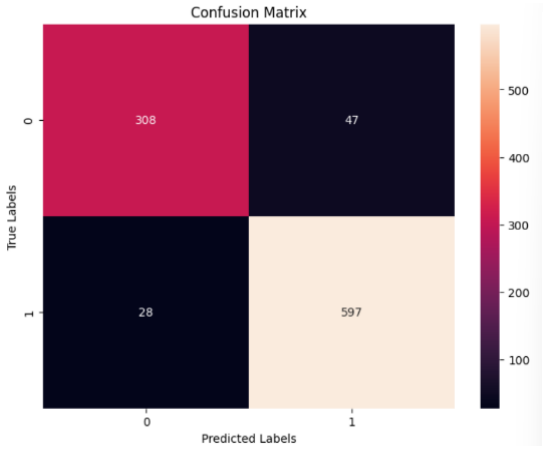

Figure 4 CM for Gaussian NB

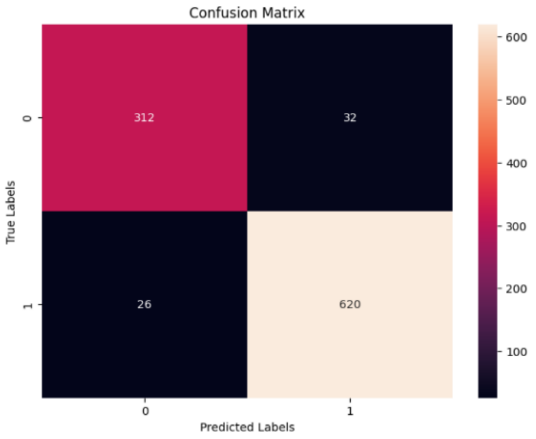

Figure 5 CM for KN Classifier

Figure 2, 3, 4 and 5 depicts the generated CM for the proposed EnS model, ET, GNB and KN model. In ML, extracted characteristics are also known as artificial nerve cell. It is a numerical form that enables for the replication of a method's outcomes. In the instance of graph division, an ensemble learning strategy is particularly beneficial. Table.2 shows CM values for basis models and proposes a Meta classifier.

Table 3: CM values for base models and proposed Meta classifier.

MODEL	TP	TN	FP	FN
ET Classifier	623	318	28	21
GNB	620	312	32	26
KN	597	308	52	33
XGB	619	310	36	25
HGB Classifier	625	311	30	24
Stacking classifier	628	325	21	16

The outcomes of our suggested ensemble classifier reveal that adopting a stacking model is the finest option, as it beats all separate classifiers, as shown in the Table 2.

Table 4: Comparative results of base models with proposed Meta classifier.

MODEL	Accuracy (%)	Precision (%)	Recall (%)	Sensitivity	Specificity
Extra Tree Classifier	95.05	95.70	96.74	0.9674	0.9191
Gaussian NB	94.14	95.09	95.98	0.9598	0.9070
KNN Classifier	91.41	92.70	94.02	0.9402	0.8676
XGB	95.16	94.23	93.12	0.9304	0.8945
Hist Gradient Boosting Classifier	94.23	91.18	92.14	0.9145	0.9547
Proposed EnS classifier	96.26	96.76	97.52	0.9752	0.9393

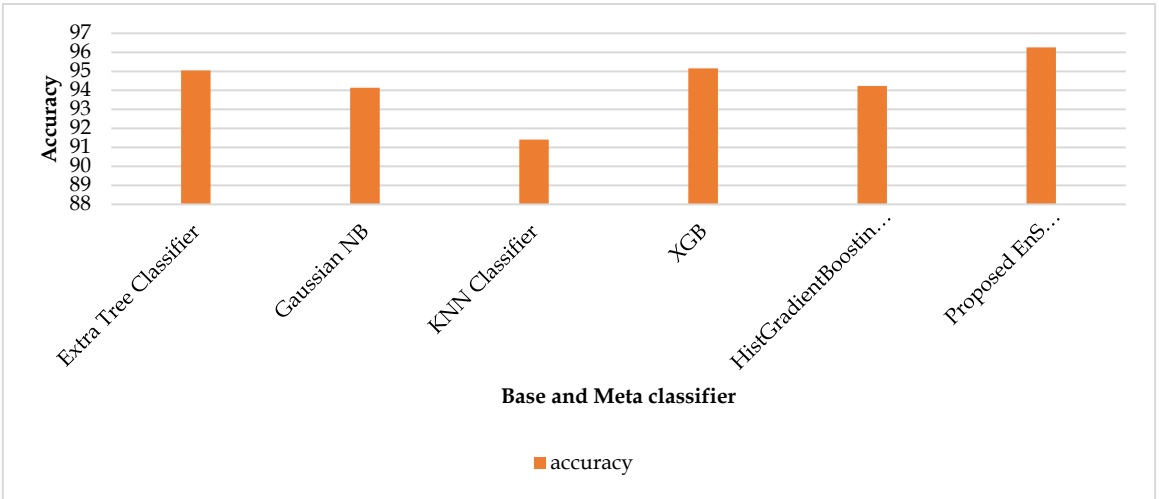

Figure 6: Performance comparison of accuracy based on Proposed Ensemble stacking model with other base models.

The proposed EnS model, ETC, Gaussian NB, KN, XGB and HGB classifier with an accuracy of 96.26%, 95.05%, 94.14%, 91.41%, 95.16% and 94.23%. In Figure 6, the proposed EnS model achieved the highest accuracy performance at 96%.

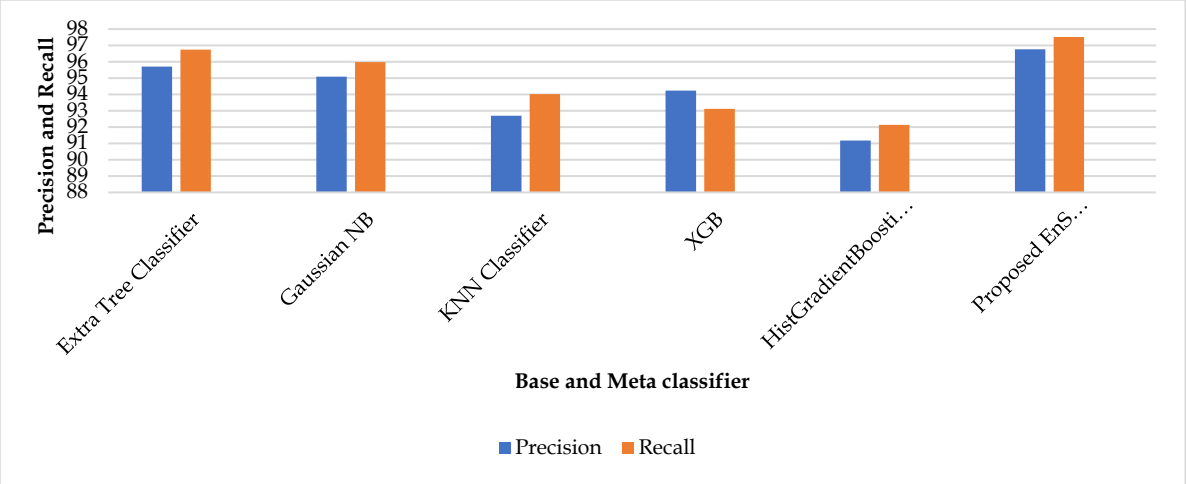

Figure 7: Performance comparison of precision and recall based on Proposed EnS model with other ensemble base models.

The precision and recall metrics are presented in Figure 7. It shows that the best result is at 96.76% for precision and 97.52% for recall which originates from the proposed EnS model. The subsequent best result for precision and recall is 95.70% and 96.74% from Gaussian NB and the nastiest performing models based on precision and recall is XGB with 94.23% and 93.12%.

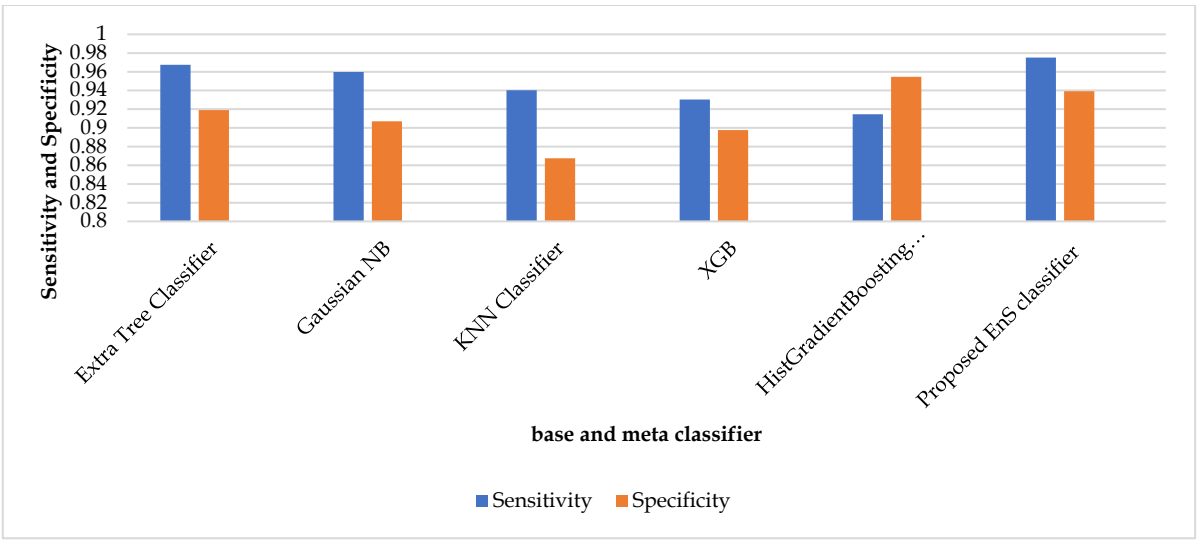

Figure 8: Performance comparison of sensitivity and specificity based on Proposed EGKH model with other ensemble base models.

The sensitivity and specificity results accomplish the assessment of the performance models in Figure 8, with the proposed EnS model having the best result with 0.9752 and 0.9393, the next highest sensitivity and specificity result is ETC with 0.9674 and 0.9191. The last two-classifier models, Gaussian NB have results of 0.9598 and 0.9070, and KN have result of 0.9402 and 0.8676 respectively.

From stacking classifier:

The performance analysis was led on a sample of 990, which included 644 malware instances and 346 benign instances. This ensemble model was able to recognise 628 samples as malicious (true positives) and 16 malware samples as benign (false negatives). Moreover, 325 benign samples were correctly labeled as legitimate (true negatives), and 21 benign samples were incorrectly flagged as malicious (false positives). These outcomes have shown that the model has a high detection rate at a low false alarm rate, thus proving that the model is practical in

differentiating between malicious and legitimate behaviour in dynamic API-based analysis.

Malware Containment Rate (Runtime)

MCR = $\frac{TP}{TP+FN} = \frac{628}{644} = 97.52\ \%$

97.52% of malicious binaries were successfully blocked after behavioral evaluation.

This is simply **Recall (TPR)**, reframed for enforcement.

Residual Runtime Risk

RR = $\frac{FN}{Total\ Malware} = \frac{16}{644} = 2.48\ \%$

These are malicious samples that executed despite sandbox-based analysis.

False Enforcement Rate

FER = $\frac{FB}{Total\ Benign} = \frac{21}{346} = 6.07\%$

This represents legitimate software incorrectly blocked after dynamic inspection.

Trust Score Computation:

Step A: Malware Probability for Allowed Files

Allowed files = Predicted benign = TN + FN = 325 + 16 = 341

Malware among allowed = FN = 16

$P_{mal\_allowed} = \frac{FN}{TN+FN} = \frac{16}{341} =$ **0.0469**

$T_{allowed} = 1 - 0.0469 = 0.9531$

So average trust of allowed files = **95.31%**

Step B: Malware Probability for Blocked Files

Blocked files = Predicted malicious = TP + FP = 628 + 21 = 649

Malware among blocked = TP = 628

$P_{mal\_blocked} = \frac{628}{649} =$ **0.9676**

Trust: $T_{blocked} = 1 - 0.9676 =$ **0.0324**

Average trust of blocked files = **3.24%**

Dynamic Trust Update

$Trust_t = \alpha\ Trust_{t-1} + (1 - \alpha)\ (Trust_{Current})$

Since $\alpha = 0.7$ and Initial $Trust_0 = 0.5$, For Allowed File (Trust_current = 0.9531)

First update: $Trust_1 = 0.7(0.5) + 0.3(0.9531) = 0.35 + 0.2859 = 0.6359$

Second update: $Trust_2 = 0.7(0.6359) + 0.3(0.9531) = 0.4451 + 0.2859 = 0.7310$

Trust gradually converges toward 0.9531.

For Blocked File (Trust_current = 0.0324)

Trust1 = 0.7(0.5) + 0.3(0.0324) = 0.35 + 0.0097 = 0.3597

Trust2 = 0.7(0.3597) + 0.3(0.0324) = 0.2518 + 0.0097 = 0.2615

Trust decays toward 0.0324.

Policy Decision Implications

The empirical trust estimation derived from the confusion matrix reveals a clear separation between two execution states. Predicted benign executables exhibit an average trust score of approximately 95.31%, whereas predicted malicious executables demonstrate a substantially lower average trust of 3.24%. This pronounced trust gap indicates that the ensemble output produces a near-bimodal trust distribution, which is desirable in Zero Trust enforcement contexts. this kind of separation makes it possible to have trusted threshold-based policy control. By properly chosen levels of trust (e.g., 0.90 to allow, 0.50 to block) the benign software is always allowed and the malicious binaries are well contained. The low number of high-trust regions and low-trust regions ensures that there is less ambiguity in the decisions of enforcement and limits the need for intermediate isolation actions. The Malware Containment Rate of 97.52% is operationally strong in blocking of malicious binaries. Meanwhile, the Residual Runtime Risk is low (2.48) as well, which makes the exposure to misclassification minimal. False Enforcement Rate is 6.07% which indicates that the business disruption can be controlled with containing strength that is not so much that it interferes with legitimate software execution.Together, these findings suggest that integrating ensemble classification outputs into a trust-based Zero Trust policy framework enables controlled risk reduction while maintaining operational feasibility.

5. CONCLUSION

In conclusion this Research develops behavioural malware detection through the combination of representation learning, ensemble intelligence, and enforceable Zero Trust control into a unified system of operation. Rather than using detection as a discrete prediction task, the suggested architecture reforms the decision pipeline by using a form of continuous validation of trust. The sequences of API calls of high dimension are converted into a topology aware manifold space, which allows the model to learn latent behavioural relationship that would be largely overlooked in more traditional feature engineering methods. It is then on this structured representation that the stacked ensemble uses to attain stable generalization in a large and changing family of malware. the main contribution of the paper is the practical implementation of zero trust principles in the detection process. The process of classification are not viewed as the final decisions but are converted into dynamic trust scores that administer the execution

privilege and containment policies. This eliminates implicit post-detection trust and presents runtime accountability to malware defense. Consequently, the system is a dynamic control system that is not only a detector but also can restrict unauthorized behaviour based on changing threat conditions. The experimental findings shows that there is uniformity in performance and high reliability of the containment in the heterogeneous samples. it establishes a scalable foundation for integrating measurable trust control into behavioural malware analysis. Future work will focus on lightweight embedding strategies and adversarial robustness to support real-time deployment scenarios.

Authors' contributions

Sasikala S: investigation, conceptualization, methodology, materials, writing, editing, experiment.
Saranya G: supervision, discussion, Validation of results.

Funding:

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of interests:

"The authors declare no competing interests."

REFERENCE

[1] Han, R.; Kim, K.; Choi, B.; Jeong, Y. A Study on Detection of Malicious Behavior Based on Host Process Data Using Machine Learning. Appl. Sci. (2023) 13, 4097. [Google Scholar] [CrossRef]
[2] Alrobaian, S.; Alshahrani, S.; Almaleh, A. Cybersecurity Awareness Assessment among Trainees of the Technical and Vocational Training Corporation. Big Data Cogn. Comput. (2023) 7, 73.
[3] J. Anderson and L. Rainie. concerns about the future of people'swell-being, (Dec 2019)
[4] AV-TEST. Malware Statistics & Trends Report. 2021. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 15 March 2023).
[5] Ferhat Ozgur Catak, Ahmet Faruk Yaz, Ogerta Elezaj and Javed Ahmed, "DL based Sequential model for malware analysis using Windows exe API Calls", PeerJ Computer Science, (2020), 21.
[6] A. Sidhardhan, S. Keerthana, J. M. Kannimoola, Weaponizing real-world applications as c2 (command and control), in: 2023 International Conference on Innovative Data Communication Technologies and Application (ICIDCA), IEEE, (2023) pp. 458–463.
[7] R. Sihwail, K. Omar, K. A. Zainol Ariffin, A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis 8 (2018) 1662. doi:10.18517/ijaseit.8.4-2.6827.
[8] J. S. Jones, M. Y. Lim, and W. J. Ray, "Challenges of static analysis for zero-day MD," *Journal of Information Security and Applications*, Feb. (2023) vol. 51, pp. 77-84, [Online]. Available: https://doi.org/10.1016/j.jisa.2023.04.004
[9] R. H. Kumar, S. Singh, and P. Sharma, "Dynamic MD: Evaluating scalability of feature extraction models," *Computers & Security*, vol. 124, p. 102874, May (2023). [Online]. Available: https://doi.org/10.1016/j.cose.2023.102874
[10] P. L. Lee, J. M. Han, and T. M. Zhu, "Challenges in MD: Datasets and real-world applicability," *IEEE Security & Privacy*, vol. 22, no. 6, pp. 32-40, Nov.-Dec. (2024). [Online]. Available: https://doi.org/10.1109/MSEC.2024.000123
[11] M. A. Ali, Z. D. Wang, and S. A. Kadri, "Improving explainability in machine learning-based cybersecurity systems," *Computers & Security*, vol. 126, p. 102850, Aug. (2023). [Online]. Available: https://doi.org/10.1016/j.cose.2023.102850
[12] A. Aslan, R. Samet, A comprehensive review on MD approaches, IEEE Access 8 (2020) 6249–6271. doi:10.1109/ACCESS. 2019.2963724.
[13] Yousuf, M.I.; Anwer, I.; Riasat, A.; Zia, K.T.; Kim, S. Windows MD based on static analysis with multiple features. PeerJ Comput. Sci. (2023), 9, e1319. [CrossRef] [PubMed]
[14] deOliveira, A.S.; Sassi, R.J. Behavioral MD using deep graph convolutional neural networks. Authorea Prepr. (2023), 1–17. [CrossRef]
[15] Network, M.D. DeleteFileA Function (fileapi.h). Available online: https://learn.microsoft.com/en-us/windows/win32/api/ f ileapi/nf-fileapi-deletefilea (accessed on 17 June 2024).
[16] Mehrabi Koushki, M.; AbuAlhaol, I.; Raju, A.D.; Zhou, Y.; Giagone, R.S.; Shengqiang, H. On building machine learning pipelines for Android malware detection: A procedural survey of practices, challenges and opportunities. Cybersecurity (2022), 5, 16.

[17] Carlin, D.; O'Kane, P.; Sezer, S. A cost analysis of machine learning using dynamic runtime opcodes for malware detection. Comput. Secur. (2019), 85, 138–155.
[18] Bhavana, G., Pudota, M. B. C., Saravanan, S., & Deepak, K. (2024, November). Improving Obfuscated Malware Classification: A Machine Learning Approach with Feature Selection and Data Balancing. In *2024 11th International Conference on Advances in Computing and Communications (ICACC)* (pp. 1-6). IEEE.
[19] Pektaş, A.; Acarman, T. Classification of malware families based on runtime behaviors. J. Inf. Secur. Appl. (2017), 37, 91–100.
[20] Suaboot, J.; Tari, Z.; Mahmood, A.; Zomaya, A.Y.; Li, W. Sub-curve HMM: A malware detection approach based on partial analysis of API call sequences. Comput. Secur. (2020), 92, 101773.
[21] Singh, J.; Singh, J. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. Inf. Softw. Technol. (2020), 121, 106273.
[22] AlAhmadi, B.A.; Martinovic, I. MalClassifier: Malware family classification using network flow sequence behaviour. In Pro ceedings of the 2018 APWG Symposium on Electronic Crime Research (eCrime), San Diego, CA, USA, 15–17 May (2018); pp. 1–13.
[23] Almushabb, R.; Ogran, R. Malware API Calls Detection Using Hybrid Logistic Regression and RNN Model. Appl. Sci. (2023), 13, 5439. https://doi.org/10.3390/ app13095439.
[24] Christofer Fellicious, Manuel Bischof, Kevin Mayer, Dorian Eikenberg, Stefan Hausotte, Hans P. Reiser, and Michael Granitzer, "MD based on API calls", arXiv:2502.12863v1 [cs.CR] 18 Feb (2025).
[25] Zhang, S.; Wu, J.; Zhang, M.; Yang, W. Dynamic Malware Analysis Based on API Sequence Semantic Fusion. Appl. Sci. (2023), 13, 6526. https://doi.org/10.3390/ app13116526.
[26] Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, G.-G.; Chen, J. Detection of Malicious Code Variants Based on DL. IEEE Trans. Ind. Inform. (2018), 14, 3187–3196.
[27] Aghakhani, H.; Gritti, F.; Mecca, F.; Lindorfer, M.; Ortolani, S.; Balzarotti, D.; Vigna, G.; Kruegel, C. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium (2020) San Diego, CA, USA, 23–26 February 2020.
[28] Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. DL Approach for Intelligent Intrusion Detection System. IEEE Access (2019), 7, 41525–41550.
[29] Jha, S.; Prashar, D.; Long, H.V.; Taniar, D. Recurrent neural network for detecting malware. Comput. Secur. (2020), 99, 102037.
[30] Gautam Karat, Jinesh M. Kannimoola, Namrata Nair, Anu Vazhayil, Sujadevi V G and Prabaharan Poornachandran, "CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection", 5th International Conference on Innovative Data Communication Technologies and Application, Procedia Computer Science 233 (2024) 492–503.
[31] Brezinski, K., & Ferens, K. (2023). Metamorphic malware and obfuscation: a survey of techniques, variants, and generation kits. *Security and Communication Networks*, *2023*(1), 8227751.
[32] Optimizing Smart Contract Security: A Cost-Sensitive Graph Neural Network Approach for Vulnerability Detection, IEEE International Conference on Electronic Systems and Intelligent Computing, ICESIC 2024 - ProceedingsConference Paper2024. DOI: 10.1109/ICESIC61777.2024.10846442.
[33] Bishwajit Prasad Gond and Durga Prasad Mohapatra, "DL-Driven Malware Classification with API Call Sequence Analysis and Concept Drift Handling", arXiv: 2502. 08679v1 [cs. LG] 12 Feb (2025).
[34] Kazım Kılıç, Ismail Atacak and Ibrahim Alper Dogru, "FABLDroid: MD based on hybrid analysis with factor analysis and broad learning methods for android applications", Engineering Science and Technology, an International Journal, (2025), https://doi.org/10.1016/j.jestch.2024.101945.
[35] Prattipati, A., Saravanan, S., & Veluchamy, S. (2024, October). Adaptive Malware Detection in Android: An Active Learning and XGBoost Approach. In *2024 5th IEEE Global Conference for Advancement in Technology (GCAT)* (pp. 1-6). IEEE.
[36] Nsikak Pius Owoh, Adejoh John, Salaheddin Hosseinzadeh and Moses Aprofin Ashawa, "MD Based on API Call Sequence Analysis: A Gated Recurrent Unit–Generative Adversarial Network Model Approach", Future Internet, (2024), DOI:10.3390/fi16100369.
[37] Ma C, Li Z, Long H, Bilal A, Liu X (2025) A malware classification method based on directed API call relationships. PLoS ONE 20(3): e0299706, https://doi.org/10.1371/ journal.pone.0299706.
[38] XueJ, WangZ, Feng R. Malicious network software detection based on API call. In: 2022 8th Annual International Conference on Network and Information Systems for Computers (ICNISC); (2022). p. 105–110.

[39] GaoH,Cheng S, Zhang W. GDroid. Android MD and classification with graph convolutional network. Comput Secur. (2021);106:102264. https://doi.org/10.1016/j.cose.2021.102264
[40] Qiang, W.; Yang, L.; Jin, H. Efficient and Robust Malware Detection Based on Control Flow Traces Using Deep Neural Networks. Comput. Secur. (2022), 122, 102871.
[41] Rosenberg, I.; Shabtai, A.; Elovici, Y.; Rokach, L. Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7–11 December (2020); pp. 611–626.
[42] Huang, W.; Stokes, J.W. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In Proceedings of the De tection of Intrusions and Malware, and Vulnerability Assessment, San Sebastián, Spain, 7–8 July (2016); pp. 399–418.
[43] Johnson, A.; Brown, R. Extracting API Sequences from Malware Samples for Enhanced Detection. Int. J. Comput. Secur. (2023), 22, 175–192.
[44] Lee, K.; Martin, M. Training Models on Secondary Datasets for Enhanced MD. J. Mach. Learn. Cybersecur. (2023), 12, 89–106.
[45] Software Composition Analysis for Proactive Threat Detection in Software Dependencies with Real-Time Security Monitoring, 2025 2nd International Conference on Computing and Data Science, ICCDS 2025Conference Paper2025. DOI: 10.1109/ICCDS64403.2025.11209681.
[46] Viswanathan, J., & Kumar, S. U. (2024, November). Zero trust security for web applications in microservice-based environments. In *2024 First International Conference on Data, Computation and Communication (ICDCC)* (pp. 488-494). IEEE.
[47] Mpasco, Mpasco/malbehavd-v1: Public datasets of malware and benign executable files (windows exe files). The dataset can be used by cybersecurity researchers focusing on the area of MD. it is suitable for training and testing both machine learning and DL algorithms. URL https://github.com/mpasco/MalbehavD-V1.