Cureus Journal of Computer Science

Open Access Original Article

Intelligent System for Malware Detection (ISMD): A Hybrid Intelligent System for Efficient Malware Detection

Received 08/21/2025
Review began 08/24/2025
Review ended 12/06/2025
Published 12/08/2025

DOI:
https://doi.org/10.7759/s44389-025-08184-z

Olorunjube J. Falana [1] , Simon Sodiya [1] , Saidat A. Onashoga [1] , Alaba O. Adejimi [1]

1. Computer Science, Federal University of Agriculture, Abeokuta, NGA

Corresponding author: Olorunjube J. Falana, falanaoj@funaab.edu.ng

Abstract

The demand for anti-malware software products has continued to grow globally due to the increasing rate of malware attacks. Most existing antimalware solutions, such as Kaspersky, Norton, and Bitdefender, rely on signature-based detection techniques. Signature-based techniques are ineffective in detecting new and unseen malware. This research proposes a hybrid-based approach named Intelligent System for Malware Detection (ISMD), which is based on a random forest classifier. As part of the approach for developing ISMD, static and dynamic analysis methods have been developed for extracting both malware and benign features. A fine-tuned feature selection technique that combined the power and efficiency of entropy and Gini gain impurity was used for selecting important features. Next, a random forest classifier model was developed to train and classify the different malware patterns. The ISMD was tested against seven other classifiers and found to have an accuracy range of 98.89%-99.05%. The ISMD was further compared with other state-of-the-art techniques using detection accuracy, training time, and prediction time. The result showed that ISMD outperformed other state-of-the-art malware detection techniques and is suitable when training and predicting time are more important in developing an anti-malware solution.

Categories: Security and Privacy, Machine Learning (ML)
Keywords: malware, static analysis, dynamic analysis, features selection, random forest classifier

Introduction

Malware is a computer file or piece of code that can infect, exploit, steal from, or attack a system owner without their knowledge [1-3]. According to the Kaspersky Security Bulletin [4], over 70% of internet users in the European Union have experienced at least one malware-class attack, and the figure is even higher in Africa. Consequently, there is a growing demand for solutions that automatically analyze and detect dangerous programs. The global antivirus software market is predicted to grow at a compound annual growth rate of 3.6%, increasing from $3.92 billion in 2021 to $4.06 billion in 2022 [4]. This trend is expected to continue, reaching $4.75 billion by 2026 at a compound annual growth rate of 4.0% [5].

Malware is constantly evolving and becoming more powerful and lethal, making data protection extremely difficult. Unless a system is sufficiently protected, it is vulnerable to contemporary computer virus threats and malware attacks [6,7]. The majority of current malware detection tools, such as Kaspersky, Norton, and Bitdefender, are typically based on a syntactic signature, which identifies byte configurations that are typical for various machine scenarios. This approach has two drawbacks. First, syntactic signatures necessitate updating the signature database whenever a previously unknown malware sample is discovered. Furthermore, there is always a gap between the presence of another instance of malicious code and the availability of a signature that can detect it [8]. Second, harmful code could be metamorphic, which means that it alters as it is repeated or spreads. To overcome these limitations, several researchers have explored the use of natural language processing techniques on malware static features such as opcodes and file sections [2,9,10]. Common natural language processing-based feature extraction methods include n-grams, which capture contiguous opcode sequences, and term frequency-inverse document frequency, which measures the relative importance of opcodes across samples. Although these techniques reduce preprocessing time, they often yield limited precision due to their inability to capture contextual and semantic relationships among instructions. Convolutional neural network, a class of deep learning algorithms, is a prospective methodology that may be used to improve recognition precision, increasing it from 92% to 99% in the identification of malware using a variety of dynamic and static characteristics [11-15]. However, convolutional neural network-based methodologies require longer pre-preparing time and adequate training data; along these lines, they require the use of powerful hardware such as graphics processing units [16].

Hence, designing an intelligent system capable of managing all categories of malware on a computer system such that it can detect unknown malware before its execution is therefore crucial in facilitating effective defense. In summary, this research addresses most of the challenges to building an intelligent malware detection system, focusing on identifying and classifying executable malware belonging to

How to cite this article
Falana O J, Sodiya S, Onashoga S A, et al. (December 08, 2025) Intelligent System for Malware Detection (ISMD): A Hybrid Intelligent System for Efficient Malware Detection. Cureus J Comput Sci 2 : es44389-025-08184-z. DOI https://doi.org/10.7759/s44389-025-08184-z

several major categories, such as trojan horse, worm, virus, spyware, ransomware, etc., each exhibiting distinct static and dynamic behavioral patterns. The followings are some of the challenges facing malware detection:

The contribution of this work includes:

· Propose a hybrid malware detection framework (ISMD) that integrates static and dynamic analysis to achieve a more comprehensive feature representation.

· Introduces an improved feature selection strategy that combines entropy and Gini impurity measures to identify the most relevant malware features and eliminate redundancy.

· Demonstrates superior performance of the proposed ISMD compared to other state-of-the-art approaches in terms of detection accuracy, training efficiency, and prediction time.

Related works

One common trait of a malware attack is the use of evasive techniques to hide its behavior in the host machine. Noor et al. [17] classified these evasion strategies into hardware-level, behavioral-level, environmental-level, and network-level categories, depending on the layer of system abstraction on which they operate. To mitigate such evasive behavior, researchers have developed various malware detection techniques, generally grouped into static, dynamic, and hybrid approaches.

In static anomaly-based detection, the detector analyses the internal structure of executable files such as API calls, opcodes, byte sequences, and control flow graphs to determine their malicious potential without executing the code. This approach offers the advantage of early detection, as it prevents the malware-carrying program from running on the host machine. Researchers have applied diverse static analysis methods, including data mining, file fingerprinting, binary analysis, and image-based feature extraction, to enhance the accuracy and scalability of malware classification.

Gaber et al. [18] introduced Alpha, a zero-day malware detection framework that integrates assembly-level static instrumentation with transformer-based modeling. Using Peekaboo, a dynamic binary instrumentation tool, the authors recorded fine-grained instruction traces to enhance the contextual representation of executable behavior. To prevent overfitting, functions common to both training and test binaries were excluded, ensuring that the model did not rely on memorized code patterns. A transformer-based model was then employed to learn the semantic and contextual embeddings of assembly instructions. While Alpha demonstrated competitive accuracy, its reliance on dynamic binary instrumentation introduces significant performance overhead, making real-time deployment against highly obfuscated or adversarial malware computationally demanding.

Data mining is the process of analyzing massive amounts of data to create new patterns. Fan et al. [19] introduced a new sequence mining approach to discover dangerous sequential patterns based on machine instruction sequences obtained from Windows portable executable (PE) files. The malicious sequential pattern-based malware detection (MSPMD) data mining framework was built using this sequence mining approach. To evaluate their approach, the authors ran a series of experiments on a sample of both malicious and benign PE files. According to the data, MSPMD had an accuracy of 95.25%, a false positive rate of 6.13%, and a detection rate of 96.17%. However, classic k-nearest neighbor (KNN), which is usually regarded as sluggish, would incur a significant computational cost when classifying new data. Shailaja et al. [20] proposed MalwareNet, an intelligent malware detection and classification framework optimized for edge computing environments. MalwareNet leverages an advanced extreme learning machine (ELM) as its core classifier, offering fast inference and reduced computational cost key advantages for deployment on or near resource-constrained edge devices. Using the Microsoft Malware Classification Dataset, the model achieved an accuracy of 99.7% and an F-measure of 99.55%. Despite these promising results, MalwareNet's generalization to new, obfuscated, or adversarial malware samples remains an open research challenge.

A Service-Oriented Mobile Malware Detection System (SMMDS) based on mining strategies was proposed by Cui et al. [21]. The strategy operates independently of the mobile device by employing a service-oriented approach, which aids in the conservation of device resources. SMMDS's architecture is made up of three key modules: real-time tasks, knowledge databases, and scheduling tasks. The real-time task aids in detecting and preventing packets travelling through a gateway from accumulating, whereas knowledge databases are utilized to store packet data, knowledge, and models for mining and detection. The authors used a MapReduce programming paradigm to implement the technique on Hadoop, a distributed computing platform. Experimental results obtained using multiple algorithms revealed that Naïve Bayes achieved a relatively low accuracy of 60%, but a false alarm rate of less than 1%. The center-distance method, on the other hand, achieved 100% accuracy but suffered from a high false alarm rate of 15%. However, because SMMDS is based only on packets, it is unable to detect malware that does not send

packets or whose packet attributes are maliciously free.

Kozik [22] presented a NetFlow-Based Malware Activity Detection approach that combined NetFlow data with an ELM classifier implemented within an Apache Spark distributed framework. NetFlow, originally developed by Cisco Systems, is a network protocol designed to collect and analyze Internet Protocol traffic information, such as source and destination IP addresses, ports, and the volume of data exchanged between hosts. Kozik's method leveraged the MapReduce programming model to scale and distribute the training process of the ELM classifier for detecting malware activity based on NetFlow features. Experimental results obtained from five datasets containing botnet activities demonstrated that the proposed methodology outperformed existing methods. It achieved a false positive rate of only 2% while successfully detecting all malicious activities on infected host. In another related work, Amini et al. [23] also employed NetFlow and clustering techniques to detect anomalies in command-and-control (C2) botnets. Their approach analyzed flow-based features such as event timings, ports, IP addresses, and packet counts. Feature selection was performed using a combination of filtering, whitelisting, and blacklisting, which helped isolate the most relevant traffic patterns for effective detection.

Mohammadian et al. [24] employed a graph-based dynamic analysis approach for malware detection. The method models program execution behavior through control flow graphs and function call graphs (FCGs), capturing rich structural relationships in binary behavior. To address computational complexity, the authors applied graph reduction techniques to eliminate redundant nodes and preserve critical behavioral features. A graph neural network was then trained for classification, while GNNExplainer provided interpretability by highlighting influential subgraphs. Although the framework demonstrated scalability, further validation is required to assess its robustness under code obfuscation, adversarial manipulation, and diverse large-scale datasets.

Han et al. [25] introduced a binary-to-image projection technique for malware that is based on feature extraction. A malicious binary file is read as a vector of 8-bit unsigned integers, which is then structured into a two-dimensional array. This array is represented graphically as a grayscale image in the range 0-255 [26]. To test the approach, the authors created 10 malware datasets varying in size from 1,000 malware descriptor vectors to 10,000 (8,410 malware and 1,590 benign binary files, with confusion at 15%). The authors conducted two sets of experiments for each of the 10 datasets to compare their local sensitive hash detection approach against an exhaustive linear search method (force method) based on the Euclidean distance similarity measure. The result shows that average recall and precision were 0.82 and 0.89, respectively. According to scientists, more effort is needed to combat countermeasures such as shifting sections of a binary or adding a large amount of redundant data.

Shabtai et al. [27] developed a behavioral-based malware detector for Android applications. The techniques continuously analyze various attributes such as CPU, memory, and battery utilization. The data was fed into several detection units, each of which used its knowledge to spot malicious behavior and provide a threat assessment based on it. To create an integrated alert, the pending threat evaluations were weighted. Shirazi et al. [28] presented a lightweight architecture for cloud infrastructure scalability. The authors used a non-parametric Cauchy function that may be adjusted recursively to create a data density-based approach. Because it does not require the storage of measurements, the technique can quickly scale as the volume of metrics expands.

Building on these developments, several researchers have explored hybrid malware detection frameworks, which integrate both static and dynamic analysis techniques to overcome the individual limitations of each approach. Huda et al. [29] suggested a hybrid architecture that employs various filters and a wrapper feature selection approach to uncover the most important malware run-time behavioral features for quick malware detection. The suggested method included knowledge from the fundamental properties of malware collected by combining discriminant, minimal redundant, and maximum relevant filters with the wrapper approach. Experimental evaluation showed that this hybrid method improved detection accuracy by approximately 7-10% compared to conventional static-only and dynamic-only models, demonstrating its ability to identify previously unseen malware samples with higher precision.

Feng et al. [30] introduced HGDetector, a hybrid Android malware detection system that combines network traffic analysis with FCG embeddings. HGDetector captures network traffic features in real time and constructs FCGs to model inter-function relationships, effectively integrating behavioral and structural information. Experimental results indicate that the combination of these two modalities enhances detection accuracy and evasion resistance. However, HGDetector faces ongoing challenges related to graph embedding efficiency, feature fusion optimization, and generalization to unseen malware variants.

Ajay Kumara and Jaidhar [31] introduced an automated multilevel malware detection system (AMMDS) that combines virtual machine introspection and memory forensic analysis techniques to locate disguised processes on a guest OS and anticipate early signs of malware execution. The AMMDS system recognizes and classifies genuine operating dangerous software based on the semantically rebuilt process view of the

guest OS. The techniques also employ machine learning algorithms on the hypervisor to classify malware.

Materials And Methods

In the study of malware detection systems, researchers have proposed a variety of analytical approaches that can be broadly grouped into signature-based [32,33] and anomaly-based [34] detection methods. Given the diverse and evolving nature of malware, there is a pressing need for an intelligent detection framework that can identify both known and unknown threats across multiple categories. To address this gap, this research proposes an ISMD, whose architecture is illustrated in Figure *1*.

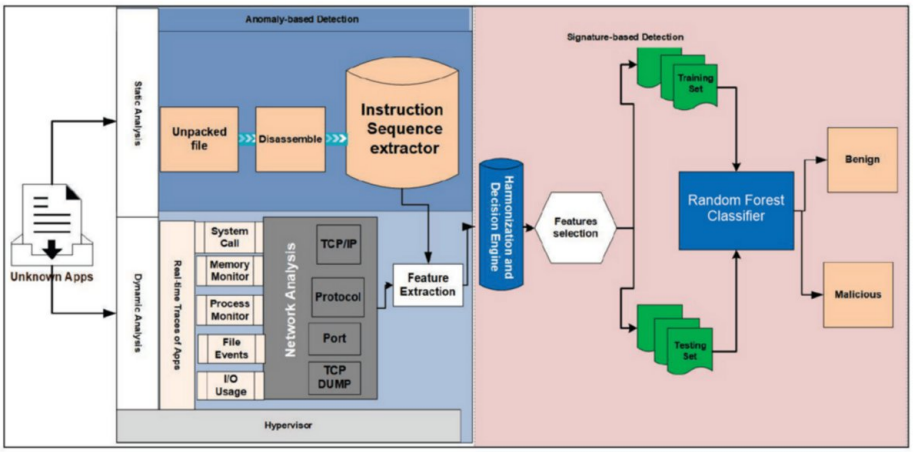

FIGURE 1: Architecture of ISMD

DLL, Dynamic Link Library; ISMD, Intelligent System for Malware Detection; TCP/IP, Transmission Control Protocol/Internet Protocol

The idea behind the ISMD is to build a robust model capable of training both the static and dynamic features for malware detection, as explained in the following sections.

Static analysis

The static analysis examined the sample without executing the codes to extract structural features such as opcodes, API calls, and n-gram sequences. This allows early identification of potential threats and reduces execution risks. However, static analysis alone may fail to detect packed, obfuscated, or polymorphic malware [18]. To address this, the ISMD includes an automated unpacking and de-obfuscation mechanism, as detailed in Figure *2*. This process begins with host checking, where the malware is tested for anti-debugging and anti-tracing behavior. If these evasive techniques are detected, the system uses controlled sandbox execution to safely observe the unpacking routine.

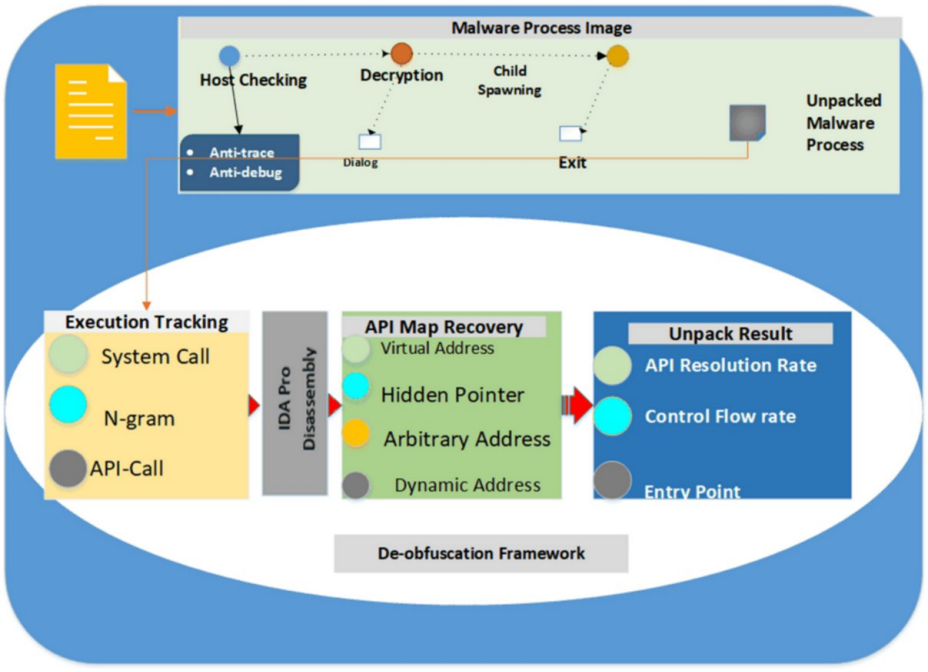

FIGURE 2: Static binary de-obfuscation analysis

API, Application Programming Interface

Execution Tracking

Execution tracking monitors the progress of unpacking the packed binaries within the controlled virtual environment. Once the process is decrypted, the malware typically invokes the NtTerminateProcess system call to export or write its payload to memory, forming the process image.

To ensure transparency and reproducibility, the unpacking process was implemented using the IDA Pro Disassembler, combined with the OllyDbg debugger for binary inspection and PEiD for packed binary identification. The framework supports Windows PE formats, including both EXE and DLL files. Following unpacking, SDoA extracts three key static features:

• System Calls: Interactions between the executing process and the operating system kernel.

• N-grams: Sequential opcode patterns representing instruction-level dependencies.

• API Calls: Function invocations within the Windows API used to perform system-level tasks.

1) System Call Representation: The system call represents the interaction between the process and the operating system, which can be expressed abstractly as a Boolean function using Equation 1.

Let $S_{\text{call}=\{x_1,x_2,x_3,\ldots,x_n\}}$ be a finite set of program variables, such as register values, memory flags, or process counters, and δ defines the state transition or behavior of the process at a given execution step.

$$\delta : S_{\text{call}} \rightarrow \{\text{true}, \text{false}\} \qquad (1)$$

2) Opcode Representation and N-grams: Let $F = \{f_1, f_2, f_3, \ldots, f_n\}$ be a dataset of disassembled file, for each $f_i \in F$.

Equation (2) defines the opcode sequence for each file:

$$S_{\text{opcode}} = \{a_1, a_2, a_3, \ldots, a_n\} \qquad (2)$$

S_{opcode} is the set of ordered opcodes in a file f_1, where $a_1 \in S_{\text{opcode}}$ for $1 \leq i \leq n$.

3) N-gram Extraction: An n-gram of length k (where $1 \leqslant k \leqslant n$) is defined as a contiguous subsequence of

opcodes drawn from the ordered sequence in Equation (3). Thus, for a given index *j*, the n-gram is:

$$S_{\text{ngram}} = (a_j, a_{j+1}, a_{j+2}, \ldots, a_{j+k-1}) \quad \text{for } 1 \leq j \leq n-k+1 \qquad (3)$$

Since every n-gram is derived from the opcode sequence of a file, it follows that:

$$S_{\text{ngram}} \subseteq S_{\text{opcode}}$$

API Map Recovery

The API map recovery process establishes a correspondence between API calls extracted from malware binaries and legitimate Windows API functions. Every executable file contains an import table directory, which records all linked Dynamic Link Libraries (DLLs) and their associated API functions. Malware authors often obfuscate or spoof these entries to disguise malicious intent. The goal of this step is, therefore, to recover a reliable mapping between the observed (possibly obfuscated) API calls and legitimate Windows APIs.

Let *D* denote the set of all API calls extracted from malware samples, and *W* represents the set of legitimate Windows APIs.

The Cartesian product of these sets defines all possible observable legitimate API associations, expressed in Equation (4).

$$D \times W = \{\, (x, y) \mid x \in D,\ y \in W \,\} \qquad (4)$$

Each API $x \in D$ may consist of a real component x_{RA} and a fake/obfuscated component x_{FA}.

Dynamic analysis

Dynamic analysis complements static analysis by examining runtime behaviors to determine the intent of a program. In the ISMD, dynamic analysis captures the following components:

1) Data Objects: ISMD employs a data-centric monitoring approach that focuses on protecting specific categories of sensitive data rather than indiscriminately tracking all files in the system. Based on the defined threat model, ISMD prioritizes monitoring user-related documents, system configuration files, and application logs that are commonly targeted by malware for exfiltr.

2) Real-time Traces of Apps: This is achieved in the ISMD by focusing on the following features: system calls, memory, processes, file events, and I/O.

3) Network Analysis: Packets are analyzed to reveal, protocol, TCP/IP, port, and other network-related information.

Harmonization and decision engine

The Harmonization and Decision Engine combines all extracted information from the static and dynamic analysis stages into a single, organized dataset. This procedure provides uniformity in feature representation while also preparing the data for categorization and learning.

Harmonization entails standardizing feature dimensions, normalizing data ranges, and resolving redundancy among static (e.g., opcodes, n-grams, API mappings) and dynamic (e.g., system calls, network traces, process behaviors) features. The end result is a full feature matrix suited for subsequent learning and detection tasks.

The Decision Engine then runs machine learning models on the harmonized dataset to assess if a given sample is malicious or benign. It supports ensemble-based decision-making procedures, which aggregate numerous model outputs to improve accuracy and robustness against adversarial variations.

Algorithm 1: Harmonization and Decision Engine

Input:
$S = \{S_1, S_2, \ldots, S_n\}$ // Set of static feature instances
$D = \{D_1, D_2, \ldots, D_m\}$ // Set of dynamic feature instances

Output: M // Matrix containing all feature vectors V

Steps:

1. Initialize an empty feature list F ← ∅

2. For each static instance S_i in S do
a. Normalize S_i to the range (0, 1)
b. Remove irrelevant or unwanted features from S_i
c. Append S_i to F
End For

3. For each dynamic instance D_j in D do
a. Convert categorical attributes in D_j to a numeric representation
b. Normalize D_j to the range (0, 1)
c. Append D_j to F
End For

4. Harmonize all feature sets in F to ensure a consistent schema

5. Construct matrix M from the harmonized feature vectors in F

6. Return M

Features selection

To determine the most discriminative characteristics, a hybrid ranking method that combines entropy and Gini impurity scores is used.

Given dataset *D* with *n* features $X = \{x_1, x_2, x_3, \ldots, x_n\}$ and class labels $C = \{c_1, c_2, c_3, \ldots, c_m\}$, the contribution of each feature to the classification is presented thus:

1) Entropy: Entropy is used to determine the level of uncertainty in the dataset and is defined in Equations 5-7.

$$H(D) = \sum_{i=1}^{m} p(c_i)\ \log_2 p(c_i) \qquad (5)$$

where, $P(c_i)$ is the probability of class c_i in *D*.

For feature x_j, the expected entropy after splitting on x_j is

$$H(D \mid x_i) = \sum_{v \in \mathrm{Values}(x_i)} \frac{|D_v|}{|D|} H(D_v) \qquad (6)$$

where, *D* is the complete training dataset; |*D*| is the number of samples in *D*; *X* is the set of *n* candidate features; *C* is the set of *m* class labels (in this study: benign and malicious); x_j is the *j*-th feature; values x_j is the set of distinct values of feature x_j; and D_v is the subset of dataset *D* where feature x_j takes value *v*.

The information gain of a feature is then

$$IG(x_j) = H(D) - H(D|x_i) \qquad (7)$$

2) Gini Impurity: Gini impurity measures how frequently a randomly selected element from the class distribution in *D* would be incorrectly classified [31]. It is calculated using Equation 8.

$$G(x_j) = 1 - \sum_{i=1}^{m} (p(c_i \mid x_j))^2 \qquad (8)$$

where, $p(c_i)$ is probability of class c_i in the entire dataset *D*; $p(c_i \mid x_j)$ is the conditional probability of class c_i among samples belonging to feature f_j; *H*(·) is an entropy function; $G(x_j)$ is the Gini impurity of feature x_j.

3) Combined Feature Score: To stabilize feature ranking and mitigate bias toward features with many

distinct values, we computed the average normalized score for each feature using Equation 9.

$$S(x_i) = \frac{IG(x_j) + (1 - G(x_j))}{2} \quad (9)$$

where, $S(x_i)$ is the combined normalized score used for ranking feature x_j.

The features were ranked in descending order of S, and the top 10 with the highest average scores were classified.

Machine learning

The essence of this part is to develop an intelligent system that is capable of learning the pattern of the data. The random forest algorithm was chosen for its stability, capacity to handle high-dimensional data, and resistance to overfitting. Random forest creates an ensemble of decision trees, each trained on a distinct subset of the data and features, and then aggregates their outputs using majority voting. This enables the model to generalize successfully across various malware behaviors and undiscovered variants.

Random Forest

The random forest classifier is a supervised ensemble learning method that constructs multiple decision trees and aggregates their predictions to improve accuracy and reduce variance. Each tree is trained on a randomly selected subset of the training data and feature space, enabling robustness against noise and overfitting. The classifier is trained on a harmonized feature matrix M, which combines the selected static and dynamic features extracted from the malware dataset. The goal of the model is to learn a mapping from feature vectors to binary labels, where benign = 0 and malicious = 1, and to predict the class of unseen samples through an ensemble voting mechanism.

1) Training Stage: During training, the random forest algorithm generates T decision trees, each grown on different bootstrap samples and feature subsets. For each tree:

i. Bootstrap Sampling: A bootstrap sample *Dt* is drawn from the training data by sampling with replacement.

ii. Random Feature Selection: At each node, a random subset of features is selected, and the optimal split is determined using a criterion such as Gini impurity or information gain. Each tree is allowed to grow to its full depth without pruning.

iii. Tree Growth: Each decision tree is grown to maximum depth without pruning, allowing the ensemble to capture diverse decision boundaries.

The decision rule for categorizing a new instance is given in Equation 10.

$$f(x) = \text{mode}\{h_1(x),\ h_2(x),\ \ldots,\ h_T(x)\} \quad (10)$$

where, $h_t(x)$ is the prediction of the t-th decision tree, and T is the total number of trees in the forest.

2) Prediction Stage: During the prediction stage, test instances are routed through trained decision trees, with the majority class being chosen as the final decision. The model also provides class probabilities, which can be used to establish thresholds for malware risk scoring or severity classification.

Algorithm 2: Random Forest Training

Inputs:

M: Training dataset (features and labels)

N_trees: The total number of trees to grow in the forest

M_features: The number of features to randomly sample at each node split

Output:

Forest: A collection (set or list) of N_trees trained decision trees

Steps:

1. Initialize Forest = ∅ (an empty set)

2. For t = 1 to N_trees do:

a. Create a bootstrap sample M_t by randomly sampling n instances from M with replacement (where n is the total number of samples in M).

b. Grow a new decision tree, h_t, using the bootstrap sample M_t.

c. To grow the tree h_t:

• At each node, randomly select M_features from the full set of features.

• Find the best split by searching only among the selected M_features using a criterion (e.g., Gini impurity).

• Recursively split the node until a stopping condition is met (e.g., the node is pure). Do not prune.

3. Add the trained tree h_t to the Forest.

4. End For

Return Forest

Algorithm 3: Random Forest Prediction

Inputs:

Forest: The trained forest from Algorithm 2.1

x_new: A single, unseen sample (feature vector)

Output:

Predicted_Class: The final class prediction (Benign or Malicious)

Steps:

1. Initialize a list of predictions (votes): Votes = []

2. For each tree h_t in Forest do:

a. Generate a prediction for x_new using the tree: vote_t = h_t(x_new)

b. Add the prediction vote_t to the Votes list.

3. End For

4. Predicted_Class = Majority Vote (mode) of the Votes list.

5. Return Predicted_Class

Results And Discussion

Implementation

In this section, each component of the ISMD and the entire detection system of ISMD are compared against a few existing methods to evaluate each portion of our framework and the entire detection system of ISMD.

Dataset description

In this experiment, 31,530 malware samples from VirusShare [35] were analyzed, and 12,346 benign

executable programs were collected from Filehippo [36], Download.com, and local program files. The benign files were scanned using Virustotal [37] to check their authenticity. VirusTotal is a free virus, malware, and URL scanning service with over 70 antivirus scanners.

Features selection and evaluation parameters

To evaluate the performance of the proposed ISMD, 77 discriminative features were collected from malware samples using both static and dynamic analysis, as shown in Table *1*. Static information such as imported API calls, opcode sequences, and section entropy was collected using Python's pefile module, while dynamic features (such as system calls, network activity, and process behaviors) were generated using the Cuckoo Sandbox environment.

To address the potential class imbalance between malware and benign samples, random undersampling and stratified k-fold cross-validation (k = 10) were used to assure equal representation during training and testing.

Furthermore, all data preprocessing, feature extraction, and labeling were done separately for the malware and benign classes before merging, reducing the possibility of data leakage. The merged dataset was divided into 70% training, 20% validation, and 10% testing sets. This strategy reduces the distribution mismatch across sources and ensures that the trained model generalizes effectively over unseen samples.

Features	Features	Features
Machine	Base of Code	Minor Subsystem Version
Size of Optional Header	Base of Data	Size of Image
Characteristics	Image Base	Size of Headers
Time Date Stamp	Section Alignment	Checksum
Major Linker Version	File Alignment	Subsystem
Minor Linker Version	Major Operating System Version	DLL Characteristics
Size of Code	Minor Operating System Version	Size of Stack Reserve
Size of Initialized Data	Major Image Version	Size of Stack Commit
Size of Uninitialized Data	Minor Image Version	Size of Heap Reserve
Address of Entry Point	Major Subsystem Version	Size of Heap Commit
Loader Flags	Maximum Entropy of Sections	Minimum Virtual Size of Sections
Number of Rva and Sizes	Mean Raw Size of Sections	Maximum Virtual Size of Sections
Number of Sections	Minimum Raw Size of Sections	Number of DLL Imports
Mean Entropy of Sections	Maximum Raw Size of Sections	Total Number of Imports
Minimum Entropy of Sections	Mean Virtual Size of Sections	Ordinal Number of Imports
Number of Resources	Mean Size of Resources	Load Configuration Size
Mean Entropy of Resources	Minimum Size of Resources	Version Information Size
Minimum Entropy of Resources	Maximum Size of Resources	Number of Exports
Maximum Entropy of Resources	API_Call_1	API_Call _2
API_Call _3	API_Call _4	API_Call _5
API_Call _6	API_Call _7	API_Call _8
API_Call _9	API_Call _10	API_Call _11
API_Call _12	API_Call _13	API_Call _14
API_Call _15	API_Call _16	API_Call _17
API_Call _18	API_Call _19	API_Call _20
API_Call _21	API_Call _22	

TABLE 1: Features extracted from PE files during static and dynamic analysis

API, Application Programming Interface; DLL, Dynamic Link Library; PE, Portable Executable

To select features with high classification potential, entropy and Gini impurity were used to compute the average weight for each feature. Figure *3* shows the computed graph of each feature extracted from the dataset. Figure *3* shows the top 10 features with high classification potential.

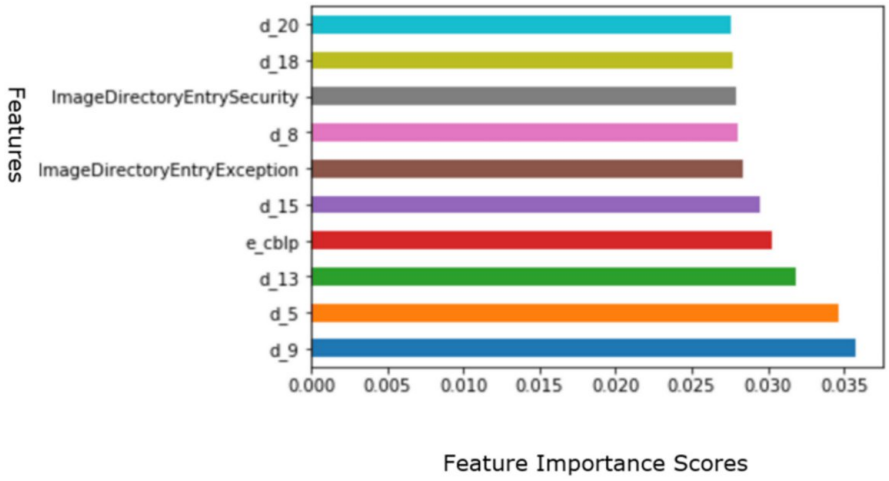

FIGURE 3: A graph of the top 10 features extracted from the dataset

To achieve accurate findings in the shortest amount of time, the top 10 features with the best discriminative power, as illustrated in Figure *3*, were chosen for the detection process. These features were chosen based on their average ranking scores, which were calculated using entropy and Gini impurity metrics to estimate the information gain and impurity reduction potential of each feature during classification.

Following feature selection, the selected features were normalized using standard scaling to achieve consistent contribution across value ranges. To ensure clarity and concise representation in the dataset, each selected feature was relabeled as FSA1-FSA10. Table *2* displays the first 14 occurrences of the preprocessed dataset, including the top 10 normalized and relabeled features and their accompanying class labels (1 = malware, 0 = benign). This separation ensures that the feature selection phase (a global ranking process) is distinct from the classification dataset (an instance-level data representation) used for model training and evaluation.

	FSA1	FSA2	FSA3	FSA4	FSA5	FSA6	FSA7	FSA8	FSA9	FSA10	Malware
0	8.228498	-4.812972	2.239394	1.473299	1.046613	0.075244	0.260836	-0.221904	-0.82999	-0.0372633	1
1	1.597827	1.707854	-1.001614	0.860487	-2.094553	0.114148	1.1117471	-0.019168	0.216647	0.598294	1
2	-0.393077	2.452353	-2.878268	0.820442	-0.325149	-0.072477	4.208709	0.408868	-1.422012	0.108424	1
3	-2.425212	1.291212	1.692576	1.942392	0.907103	-1.902121	-0.696097	0.678140	0.941066	3.305872	1
4	-4.471388	-4.632926	1.172135	-0.950441	-1.010478	-0.793234	-0.047379	1.119565	-0.381329	-1.487664	1
5	-0.325285	0.871450	-1.713911	1.001469	0.531149	1.78866	-0.458134	-0.596259	0.848543	0.013088	1
6	3.265927	-1.090190	-3.69053	-1.630293	-1.350524	-0.593330	0.875107	-1.492228	2.275002	2.186324	1
7	-1.975468	1.687247	-0.367823	0.323784	1.360376	1.618618	-2.369179	1.816655	0.2732	-1.144293	1
8	-1.262351	0.473830	-0.097628	-0.892930	3.223804	4.181502	0.974977	1.017491	2.120900	-3.659074	1
9	1.331982	1.056643	-0.369730	0.561008	1.215921	2.716275	0.809213	0.162616	2.981812	-346876	1
10	-1.748762	1.982573	0.127208	-0.525117	0.828239	-0.296971	-0.542187	-1.1036691	-2.006082	-0.232099	1
11	-1.771895	1.146906	-o.665789	1.328269	-1.505305	-1.644378	-0.476984	0.219351	-0.64129	-0.353791	1
12	-1.479588	3.842983	-0.554444	0.842193	2.598656	-0.196328	0.592082	-1.325657	-1.796589	-0.349711	1
13	-4.811720	-55.705116	0.390276	-1.275454	1.274679	-4.468180	-1.011023	1.392553	-0.382721	-1.918972	1
14	-4.811720	-5.705116	0.390276	-1.275454	1.274679	-4.468180	-1.011023	1.392553	-0.382721	-1.918972	1

TABLE 2: The 10 features selected

The ISMD evaluation parameters were based on measures like accuracy, precision, recall, and F1-score. As illustrated in Equations 10-13, these parameters serve as a benchmark for comparing the ISMD to previous approaches. For the performance study, the confusion matrix, area under the curve (AUC), and receiver operating characteristic (ROC) curve were used.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$F_{measure} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{14}$$

where, TP denotes correctly classified malware files; FP denotes benign files misclassified as malware; FN denotes the number of malware misclassified as benign files; and TN denotes correctly classified benign file cases.

Accuracy is one of the most crucial characteristics for users of a malware detection system; however, it is insufficient for our design objective because true positives and false positives are mixed, and accuracy does not provide relevant marginal interpretations. As a result, this study prioritizes precision, recall, F1-score, and AUC-ROC as major evaluation measures.

To be more specific, the series of best models is obtained through the following experiments:

• Experiment 1: Examining the effectiveness of the feature extraction process.

• Experiment 2: Run ISMD with seven different classifiers to find the best classifier.

ISMD was compared to different baseline classifiers, including Support Vector Machine (SVM), KNN,

Decision Tree, Naïve Bayes, Bagging, Extra Tree, and Gradient Boosting. Table 3 presents a summary of the outcomes.

Classifier		Accuracy (%)	Precision	Recall	F1-score	Training Time (s)	Testing Time (s)
Nonlinear models classifier							
SVM	Benign	97.57	0	0	0	1.084	0.222
	Malware		0.98	1	0.99		
KNN	Benign	98.39	0.88	0.39	0.54	1.084	0.691
	Malware		0.98	1	0.99		
Decision Tree	Benign	97.53	0.49	0.51	0.5	1.084	0.002
	Malware		0.99	0.99	0.99		
Naïve Bayes	Benign	93.82	0.12	0.25	0.16	0.028	0.002
	Malware		0.98	0.96	0.97		
Ensemble models classifier							
Bagging	Benign	98.59	0.97	0.43	0.6	0.166	0.028
	Malware		0.99	1	0.99		
Extra Tree	Benign	98.55	0.98	0.41	0.58	0.158	1.467
	Malware		0.99	1	0.99		
Gradient Boosting	Benign	98.19	0.86	0.3	0.45	2.386	0.014
	Malware		0.98	1	0.99		
Proposed method with 10 features set							
ISMDS	Benign	98.98	1	0.68	0.72	1.043	0.129
	Malware		0.99	1	0.99		
Proposed method with 77 features set							
ISMDS_1	Benign	99.05	0.96	0.64	0.77	26.625	0.425
	Malware		0.99	1	1		

TABLE 3: Performance analysis of our ISMD

ISMD, Intelligent System for Malware Detection; KNN, K-Nearest Neighbors; SVM, Support Vector Machine

For each classifier, metrics are reported per class (benign and malware) to highlight differences in detection capability across categories. While malware cases are recognized with near-perfect precision and recall across all models, benign files have significantly lower precision and recall values, demonstrating a bias towards malware, which can be attributed to malware samples outnumbering benign executables.

Comparative analysis

The proposed ISMD had the best overall performance across most measures. Specifically, the model achieved 98.98% accuracy and an F1-score of 0.72 for benign detection, as well as an F1-score of 0.99 for malware detection, utilizing the top 10 selected characteristics. The version trained with all 77 characteristics (ISMDS_1) has slightly greater accuracy (99.05%) and a moderate improvement in F1-score (0.77). However, this improvement came at a significant computational cost: training time increased from 1.043 seconds to 26.625 seconds.

This result shows that the feature selection technique considerably improves efficiency without sacrificing performance. As a result, the top 10 features derived from Gini impurity and entropy-based selection create a more computationally efficient and interpretable model that is suitable for real-world

applications. The relatively poor benign-class recall across models suggests that future optimization should focus on lowering false positives, possibly by using more diversified benign samples and adaptive decision criteria.

Figures 4-5 compare the confusion matrix between ISMD and other classifiers. The results show that ISMD improves detection accuracy, with a higher rate of correctly identified malware cases.

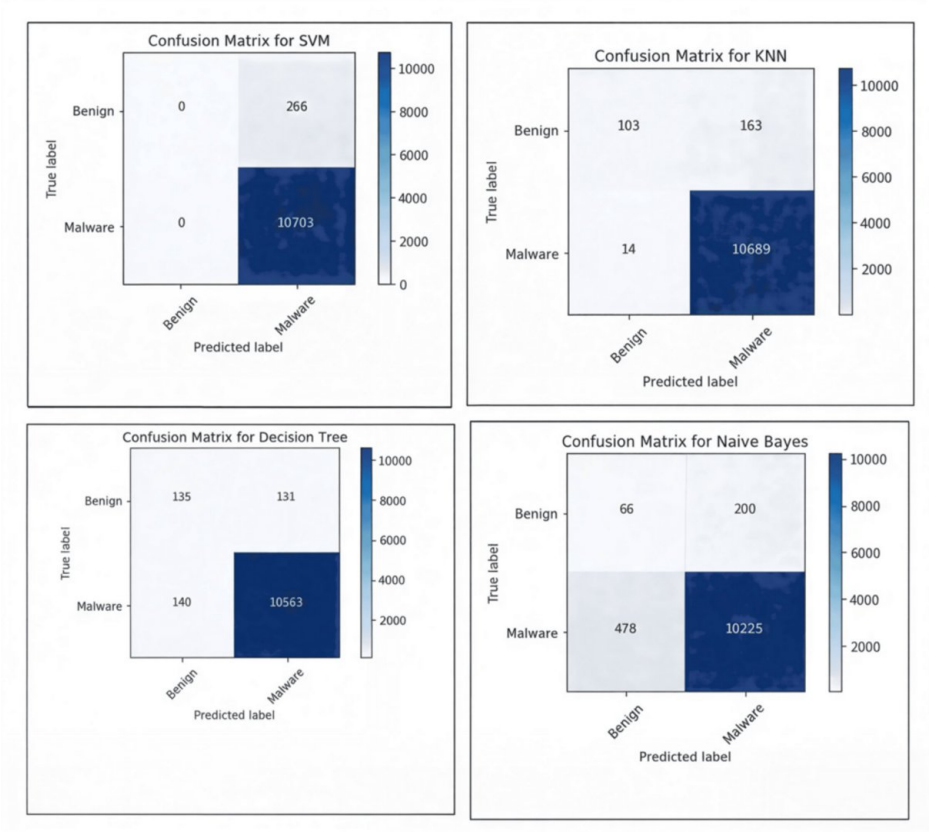

FIGURE 4: Comparison of confusion matrix with other classifiers

KNN, K-Nearest Neighbors; SVM, Support Vector Machine

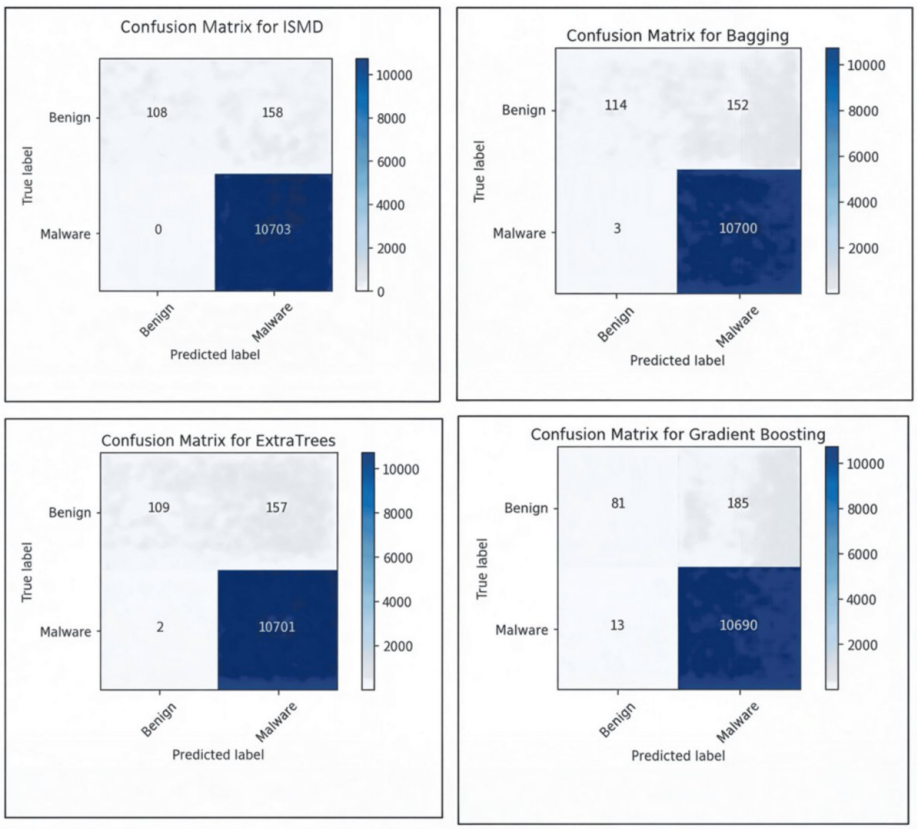

FIGURE 5: Comparison of confusion matrix of ISMD with other classifiers

ISMD, Intelligent System for Malware Detection

Figures *6*-*7* show the ROC curves for ISMD and the comparable classifiers. The ROC curve compares the True Positive Rate (TPR) against the False Positive Rate (FPR) at different thresholds. The ideal classifier would appear in the upper-left corner of the plot, indicating flawless detection (TPR = 1) and no false alarms (FPR = 0). Although this ideal performance is rarely obtained, the AUC offers a quick overview of each classifier's discriminative ability - higher values indicate better overall performance.

As shown in Figure *7*, the AUC values for the classifiers are as follows: SVM = 0.50, KNN = 0.70, Decision Tree = 0.75, Naïve Bayes = 0.62, ISMD = 0.74, Bagging = 0.74, Extra Tree = 0.74, and Gradient Boosting = 0.68.

Decision tree- and ISMD-based models had the highest AUC scores (0.74-0.75), showing strong discrimination between benign and malicious classes. In comparison, SVM yielded an AUC of 0.50, which is similar to random guessing, implying that generalization to unseen samples is limited. Although ISMD does not achieve complete separation, its high AUC value indicates a favorable trade-off between detection accuracy and false-positive reduction, demonstrating that the chosen features improve the resilience of malware detection when compared to most baseline models.

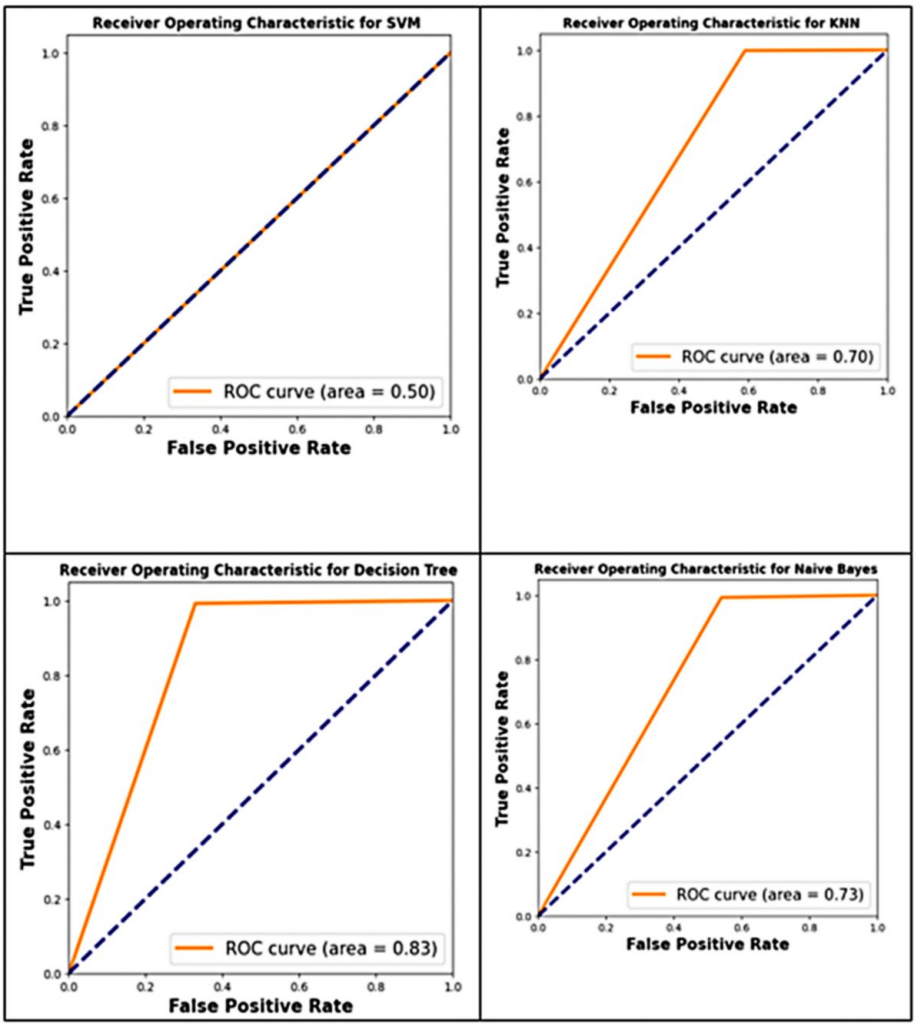

FIGURE 6: ROC comparison with other classifiers

KNN, K-Nearest Neighbors; ROC, Receiver Operating Characteristic curve; SVM, Support Vector Machine

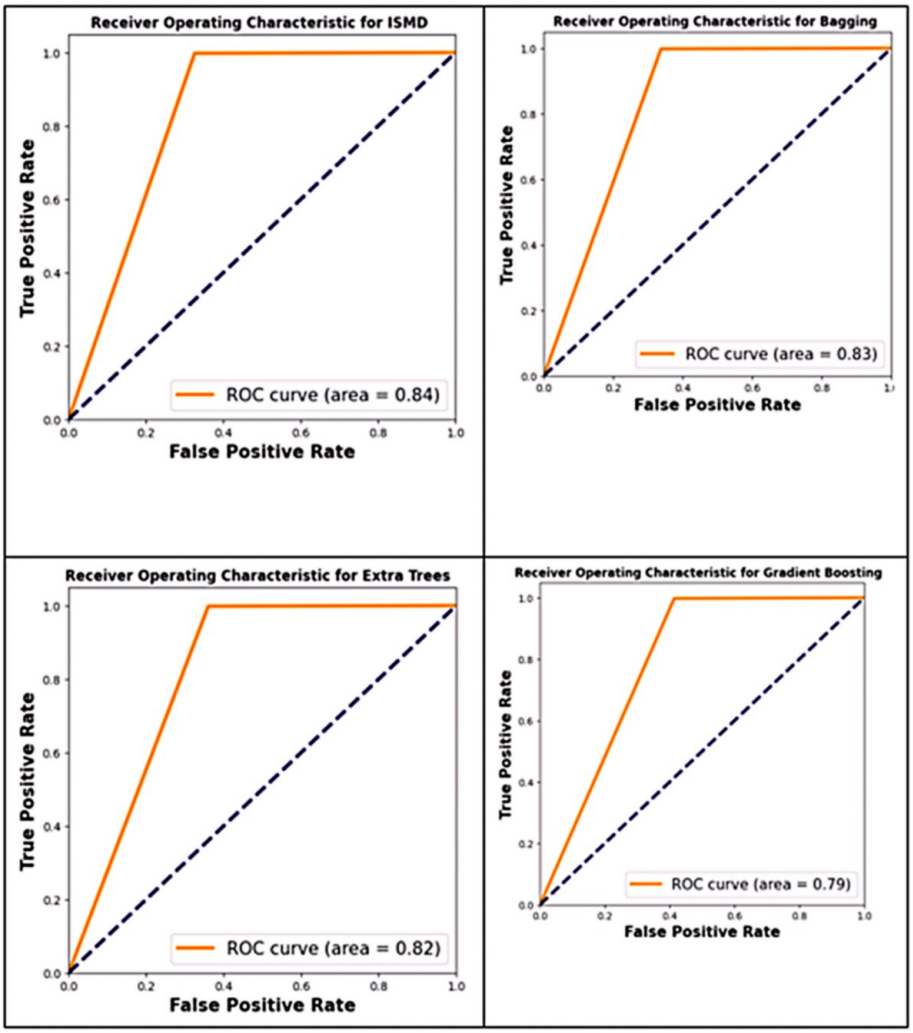

FIGURE 7: ROC comparison of ISMD with another classifier

ISMD, Intelligent System for Malware Detection; ROC, Receiver Operating Characteristic curve

Comparison with existing work

Table *4* compares the proposed ISMD with selected existing malware detection methods based on opcode, instruction sequence, and memory analysis. These methods were chosen because they are well-established and have proven strong detection performance in previous research.

The dataset sizes, evaluation measures, and experimental setups differ among the works cited. For example, Yadav [6] and Fan et al. [19] used smaller datasets with clustering or sequence-mining algorithms, whereas Nissim et al. [7] used memory dump data analyzed with TPR, FPR, and AUC. ISMD, on the other hand, was trained on a bigger dataset of 43,386 samples utilizing API-call characteristics, and its performance was evaluated using accuracy, precision, recall, F1-score, and ROC-AUC.

While Nissim and Lahav [6] reported an AUC of 1.0 on a controlled dataset, ISMD obtained an AUC of 0.74 on a more diverse and realistic dataset (Figures *6*-*7*), demonstrating its capacity to efficiently balance TPR and FPR under real-world settings. Despite differences in dataset design, ISMD achieved a competitive accuracy of 98.98%, which is equivalent to or slightly higher than other existing approaches. Although Kakisim et al. [38] obtained comparable accuracy with opcode-based features, ISMD offers faster prediction (0.129 s) and better interpretability due to random forest-based feature selection.

Approach	Dataset	Feature Type	Method	Training Time	Prediction Time (s)	Accuracy (%)
Yadav [6]	Not reported	Not reported	Weighted Fuzzy K-means clustering	Not reported	2045	92.45
Kakisim et al. [38]	10,942	Opcode	Co-opcode graph	48.42	19.12	98.9
Fan et al. [19]	10,307	Instruction sequences	Sequence mining algorithm	Not reported	Not reported	95.25
Nissim et al. [7]	Not reported	Memory dumps	MinHash method	Not reported	Not reported	TPR = 0.958, FPR = 0, AUC = 1
ISMD	43,386	API call, and	Hybrid static–dynamic analytical method	1.043	0.129	98.98

TABLE 4: Comparison of ISMD with existing work

API, Application Programming Interface; AUC, Area Under the Curve; FPR, False Positive Rate; ISMD, Intelligent System for Malware Detection; TPR, True Positive Rate

Conclusions

This study presented an ISMD system that combines static and dynamic analysis to support more reliable malware detection. The approach employed entropy and Gini impurity to guide lightweight feature evaluation prior to classification using a random forest model. The experimental evaluation demonstrated that the selected hybrid features provided a coherent representation of malware behavior, enabling ISMD to achieve stable performance across multiple metrics, including accuracy, precision, recall, F1-score, and ROC-AUC. While ISMD achieved competitive or slightly improved accuracy and AUC compared with several conventional and ensemble classifiers, the results also revealed a performance bias on benign samples. This highlights the need to further address class imbalance and improve the model's generalization capability. Comparisons with existing studies suggest that ISMD performs within the range of reported state-of-the-art methods; however, these observations must be interpreted cautiously due to differences in datasets, feature sets, and evaluation protocols, which limit direct comparative validity. Overall, ISMD offers a practical framework for integrating behavioral and statistical indicators in malware identification. The approach demonstrates efficient use of features and remains computationally manageable for moderately sized datasets, providing a foundation for incremental improvements rather than definitive superiority.

Future work will focus on improving class balance, evaluating robustness against previously unseen malware families, and reproducing existing baselines under identical experimental conditions to enable fairer benchmarking. Additional extensions will include supporting the detection of Android, IoT, and cloud-based malware and enhancing overall performance through adaptive hybrid learning and more discriminative feature-selection strategies.

Appendices

Data Availability

The datasets and code generated and/or analyzed during this study are included in this article or are available from the corresponding author upon reasonable request.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Olorunjube J. Falana, Saidat A. Onashoga , Alaba O. Adejimi, Simon Sodiya

Acquisition, analysis, or interpretation of data: Olorunjube J. Falana, Simon Sodiya

Drafting of the manuscript: Olorunjube J. Falana, Alaba O. Adejimi, Simon Sodiya

Critical review of the manuscript for important intellectual content: Olorunjube J. Falana, Saidat A. Onashoga , Alaba O. Adejimi, Simon Sodiya

Supervision: Olorunjube J. Falana, Simon Sodiya

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

Acknowledgements

The dataset is available on:
https://drive.google.com/file/d/1smSfFERalR03UpybMvY0xpUI6BmXQTFR/view?usp=sharing

References

1. Potter B, Day D: The effectiveness of anti-malware tools. Computer Fraud and Security. 2009, 2009:12-13. 10.1016/s1361-3723(09)70033-8
2. Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG: Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences. 2013, 231:64-82. 10.1016/j.ins.2011.08.020
3. Falana OJ, Sodiya AS, Onashoga SA, Badmus BS: Mal-Detect: An intelligent visualization approach for malware detection. Journal of King Saud University Computer and Information Sciences. 2022, 34:1968-1983. 10.1016/j.jksuci.2022.02.026
4. Kaspersky: Kaspersky Security Bulletin 2020-2021. EU statistics. (2021). Accessed: August 8, 2025: https://securelist.com/kaspersky-security-bulletin-2020-2021-eu-statistics/102335/.
5. Impulsec: Antivirus Statistics 2025: Positive Trends in Cybersecurity. (2025). Accessed: August 8, 2025: https://impulsec.com/antivirus-software/antivirus-statistics/.
6. Yadav RM: Effective analysis of malware detection in cloud computing. Computers & Security. 2019, 83:14-21. 10.1016/j.cose.2018.12.005
7. Nissim N, Lahav O, Cohen A, Elovici Y, Rokach L: Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud. Computers & Security. 2019, 87:101590. 10.1016/j.cose.2019.101590
8. Li J, Li Q, Zhou S, Yao Y, Ou J: A review on signature-based detection for network threats. 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), Guangzhou, China. 2017, 1117-1121. 10.1109/ICCSN.2017.8230284
9. Kang B, Yerima SY, Mclaughlin K, Sezer S: N-opcode analysis for android malware classification and categorization. 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security), London, UK. 2016, 1-7. 10.1109/CyberSecPODS.2016.7502343
10. Xu L, Zhang D, Alvarez MA, Morales JA, Ma X, Cavazos J: Dynamic android malware classification using graph-based representations. 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud), Beijing, China. 2016, 220-231. 10.1109/CSCloud.2016.27
11. Jahromi AN, Hashemi S, Dehghantanha A, Choo K-KR, Karimipour H, Newton DE, Parizi RM: An improved two-hidden-layer extreme learning machine for malware hunting. Computers & Security. 2020, 89:101655. 10.1016/j.cose.2019.101655
12. Ni S, Qian Q, Zhang R: Malware identification using visualization images and deep learning. Computers & Security. 2018, 77:871-885. 10.1016/j.cose.2018.04.005
13. Karbab EB, Debbabi M, Derhab A, Mouheb D: MalDozer: Automatic framework for android malware detection using deep learning. Digital Investigation. 2018, 24:S48-S59. 10.1016/j.diin.2018.01.007
14. Yakura H, Shinozaki S, Nishimura R, Oyama Y, Sakuma J: Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. CODASPY '18: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy. 2018, 127-134. 10.1145/3176258.3176335
15. Cui Z, Xue F, Cai X, Cao Y, Wang G-g, Chen J: Detection of malicious code variants based on deep learning. IEEE Transactions on Industrial Informatics. 2018, 14:3187-3196. 10.1109/tii.2018.2822680
16. Athiwaratkun B, Stokes JW: Malware classification with LSTM and GRU language models and a character-level CNN. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA. 2017, 2482-2486. 10.1109/ICASSP.2017.7952603
17. Noor M, Abbas H, Shahid WB: Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. Journal of Network and Computer Applications. 2018, 103:249-261. 10.1016/j.jnca.2017.10.004
18. Gaber M, Ahmed M, Janicke H: Zero day malware detection with Alpha: Fast DBI with Transformer models for real world application. Computers and Electrical Engineering. 2025, 128:110751. 10.1016/j.compeleceng.2025.110751
19. Fan Y, Ye Y, Chen L: Malicious sequential pattern mining for automatic malware detection. Expert Systems with Applications. 2016, 52:16-25. 10.1016/j.eswa.2016.01.002
20. Shailaja P, Jahan T, Sharmila K, Bharath Siva Varma P, Arra S, Kumar PM: MalwareNet: an intelligent malware detection and classification using advanced extreme leaning machine in edge computing environment.

Egyptian Informatics Journal. 2025, 31:100714. 10.1016/j.eij.2025.100714
21. Cui B, Jin H, Carullo G, Liu Z: Service-oriented mobile malware detection system based on mining strategies. Pervasive and Mobile Computing. 2015, 24:101-116. 10.1016/j.pmcj.2015.06.006
22. Kozik R: Distributing extreme learning machines with Apache Spark for NetFlow-based malware activity detection. Pattern Recognition Letters. 2018, 101:14-20. 10.1016/j.patrec.2017.11.004
23. Amini P, Azmi R, Araghizadeh MA: Botnet detection using NetFlow and clustering. Advances in Computer Science: An International Journal. 2014, 3:139-149.
24. Mohammadian H, Higgins G, Ansong S, Razavi-Far R, Ghorbani AA: Explainable malware detection through integrated graph reduction and learning techniques. Big Data Research. 2025, 41:100555. 10.1016/j.bdr.2025.100555
25. Han X, Sun J, Qu W, et al.: Distributed malware detection based on binary file features in cloud computing environment. in Control and Decision Conference (2014 CCDC. The 26th Chinese.. 2014, 10.1109/CCDC.2014.6852896
26. Nataraj L, Karthikeyan S, Iacob G, et al.: Malware images: visualization and automatic classification. in Proceedings of the 8th international symposium on visualization for cyber security.. 2011, 10.1145/2016904.2016908
27. Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y: "Andromaly": a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems. 2012, 38:161-190. 10.1007/s10844-010-0148-x
28. Shirazi SN, Simpson S, Gouglidis A, Mauthe A, Hutchison D: Anomaly detection in the cloud using data density. 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA. 2016, 616-623. 10.1109/CLOUD.2016.0087
29. Huda S, Islam R, Abawajy J, Yearwood J, Hassan MM, Fortino G: A hybrid-multi filter-wrapper framework to identify run-time behaviour for fast malware detection. Future Generation Computer Systems. 2018, 83:193-207. 10.1016/j.future.2017.12.037
30. Feng J, Shen L, Chen Z, Lei Y, Li H: HGDetector: A hybrid Android malware detection method using network traffic and Function call graph. Alexandria Engineering Journal. 2025, 114:30-45. 10.1016/j.aej.2024.11.068
31. Ajay Kumara MA, Jaidhar CD: Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM. Future Generation Computer Systems. 2018, 79:431-446. 10.1016/j.future.2017.06.002
32. Ye Y, Wang D, Li T, Ye D, Jiang Q: An intelligent PE-malware detection system based on association mining. Journal in Computer Virology. 2008, 4:323-334. 10.1007/s11416-008-0082-4
33. Bidoki SM, Jalili S, Tajoddin A: PbMMD: A novel policy based multi-process malware detection. Engineering Applications of Artificial Intelligence. 2017, 60:57-70. 10.1016/j.engappai.2016.12.008
34. Ding Y, Xia X, Chen S, Li Y: A malware detection method based on family behavior graph. Computers & Security. 2018, 73:73-86. 10.1016/j.cose.2017.10.007
35. VirusShare: Because Sharing is Caring. (2024). Accessed: August 8, 2025: https://virusshare.com/.
36. Filehippo: The latest versions of the BEST & SAFE software. (2024). Accessed: August 8, 2025: https://filehippo.com/.
37. Virustotal: Analyse suspicious files, domains, IPs and URLs to detect malware. (2024). Accessed: August 8, 2025: https://www.virustotal.com/gui/.
38. Kakisim AG, Nar M, Sogukpinar I: Metamorphic malware identification using engine-specific patterns based on co-opcode graphs. Computer Standards & Interfaces. 2020, 71:103443. 10.1016/j.csi.2020.103443