Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 19 No. 10 (2025) | OPEN ACCESS

https://doi.org/10.3991/ijim.v19i10.51319

PAPER

Elevating Mobile Security: An Ensemble Approach for Enhanced Malware Detection with M-iForest

Orieb AbuAlghanam[1](✉), Hadeel Alazzam[2], Mohammad Qatawneh[3,4], Moutaz Alazab[5], Mohammad Al Sharaiah[4]

[1]Department of Computer Science, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

[2]Department of Information Technology, Yarmouk University, Irbid, Jordan

[3]Department of Networks and Cybersecurity, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan

[4]Department of Information Technology, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

[5]Department of Intelligence Systems, Al-Balqa Applied University, Al-Salt, Jordan

o.abualganam@ju.edu.jo

ABSTRACT

The rapid advancement of technologies and the widespread use of smartphones have given rise to new malware threats. However, the sophisticated techniques adopted by malware creators have significantly diminished the effectiveness of traditional security measures, including signature-based detection and antivirus tools solutions ineffective. To address this issue, current malware detection methods rely on extracting malware features and analyzing them using static, dynamic, or hybrid techniques. In this paper, an innovative fusion Android malware detection system has been proposed. The fusion system is based on two parallel subsystems working together. The first subsystem is trained on benign-labeled applications, while the other one focuses on malware-labeled applications. Each subsystem leverages an ensemble approach, combining one class support vector machine (OC-SVM), local outlier factor (LOF), and a modified isolation forest (M-iForest) classifier. The evaluation has been done using two benchmark Android malware datasets, which are DREBIN and CICAndMal2017. The proposed system achieves an accuracy rate of 97.05% and an F-score of 95.87% for the DREBIN dataset. Similarly, for the CICAndMal2017 dataset, it attains an accuracy rate of 99.01% and an impressive F-score of 96.58%. The proposed approach outperforms several existing methods that use the same dataset in terms of accuracy, F-score, and false-positive rate (FPR).

KEYWORDS

CICAndMal2017 dataset, DREBIN dataset, fusion system, malware detection

1 INTRODUCTION

These days, smartphones and tablets have become an integral part of our lives. They're not just for staying in touch with others—they're also essential tools in areas like education, healthcare, and online shopping. But as their use has grown, so have the risks. Cyberattacks targeting mobile devices and apps are on the rise, largely due to the sensitive information we store on them, such as bank details, emails, and personal data [1].

AbuAlghanam, O., Alazzam, H., Qatawneh, M., Alazab, M., Al Sharaiah, M. (2025). Elevating Mobile Security: An Ensemble Approach for Enhanced Malware Detection with M-iForest. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(10), pp. 129–151. https://doi.org/10.3991/ijim.v19i10.51319

Article submitted 2024-07-23. Revision uploaded 2025-02-19. Final acceptance 2025-02-27.

One of the biggest concerns is the vast number of apps available, especially on Android. Since Android dominates the mobile market with over 70% of users, it has become a prime target for malware developers [2] [3]. As more people rely on smartphones and tablets, hackers have found new ways to exploit these devices, stealing personal information and causing harm.

To tackle this problem, stronger security measures are needed to detect malicious apps, vulnerabilities, and attacks—without compromising user privacy. Hackers are becoming more sophisticated, using new techniques to hide malicious code, making it harder for traditional antivirus tools to keep up. That's why there is a growing demand for more advanced malware detection techniques [4] [5].

Researchers have proposed various methods for detecting malicious apps, with machine learning (ML) and artificial intelligence (AI) leading the way, especially on Android systems [6]. These technologies have been particularly effective in classifying malware into families and identifying new variants, helping to differentiate between safe and harmful apps [7].

Another promising approach is using ML to develop intelligent malware detection systems. These systems can identify new, complex malware patterns that traditional tools might miss. For example, static code analysis—a technique that examines an app's code without executing it—has become a popular choice. It's efficient and avoids the high computational costs of running apps in a sandbox environment [8].

When analyzing apps for security risks, there are two main methods: static and dynamic analysis. Static analysis examines the app's code to identify potential threats, while dynamic analysis runs the app to observe its behavior. Although dynamic analysis can be more thorough, it's also more resource-intensive. Static analysis, on the other hand, is faster and more practical, especially when dealing with large numbers of apps or obfuscated code [9], [10], and [11].

1.1 Research motivation

The landscape of malware detection, initially designed for traditional computers, presents a daunting challenge when extended to mobile devices. Traditional malware detection tools and antivirus software primarily rely on signature-based detection, which is effective for known threats but struggles with identifying emerging, unknown threats [12]. Furthermore, mobile devices often operate under resource constraints, including limited computational power and storage capacity, which pose challenges for executing resource-intensive malware detection algorithms [13]. The sheer volume of data generated by mobile devices further compounds the challenge, as traditional detection systems may struggle to process and analyze this data in real-time. Additionally, mobile devices use unique communication methods and data formats compared to traditional computers, demanding specialized techniques for identifying unusual or suspicious activity. Moreover, physical attacks are common on mobile devices, including interference or damage, which traditional malware detection systems may not detect. These challenges highlight the need for novel methods to improve malware detection in the continuously advancing field of mobile security.

1.2 Paper contributions

This paper presents a model for detecting malware using an ensemble approach that develops a modified isolation forest (M-iForest) approach. Moreover, two popular benchmark datasets have been used to assess the model's performance, with metrics such as F-score, accuracy, and false-positive rate (FPR).The main contributions of the paper are as follows:

1. Ensemble approach: The paper introduces an ensemble approach based on a modified version of the isolation forest (iForest), local outlier factor (LOF), and OCSVM which are considered as one-class classifiers.
2. Novel malware detection system: The paper proposes a new and innovative malware detection system.
3. Comprehensive evaluation: The proposed malware detection system is thoroughly analyzed and compared with state-of-the-art methods. Evaluation metrics include F-score, precision, accuracy, FPR, and recall, using the CICAndMal2017 and DREBIN datasets.

1.3 Paper structure

The structure of the paper is organized as follows:

Section 2 presents an overview of various malware detection systems employing the DREBIN and CICAndMal2017 datasets. Following this, Section 3 provides a concise background on the three one-class classifiers used within the ensemble approach. In Section 4, detailed information about the proposed anomaly-based malware detection system is provided. The experimental results are outlined in Section 5. Finally, Section 6 concludes the paper.

2 RELATED WORK

In this section, an overview of several malware detection systems that have used different malware datasets is presented in terms of the static and dynamic feature selection and the model that has been used.

Recently, several malware detection methods have been proposed, each differing in the techniques they use and the malware datasets they employed. Some methods utilize AI, while others rely on ML algorithms.

In [14] A framework for malware detection has been proposed, evaluating several ML algorithms, including random forest (RF), J48, decision tree (DT), k-nearest neighbors (KNN), and support vector machine (SVM). The approach focuses on classifying adware malware and benign instances, utilizing the CICMalDroid 2020 dataset for evaluation. The RF algorithm achieves a precision of 98.75% and an accuracy of 98.6%, demonstrating its effectiveness.

Reducing the dimensionality of features in malware data is crucial for several reasons, such as improving performance and decreasing the time required for decision-making. In [15], a feature selection algorithm was employed to reduce the features of the DREBIN, CICAndMal2017, AM, and AMSF datasets for evaluating their model. Along with feature selection, several ML algorithms, including RF, KNN, naive bayes (NB), and SVM, were utilized.

In [16] and [17] both propose applications for detecting malware in APK files using AI and deep learning techniques. The approach in [16] specifically targets SQL injection, while [17] employs a convolutional neural network (CNN) in their models, both with and without transformer layers.

Static and dynamic feature selection are two approaches used in malware analysis to identify and classify malicious software. These approaches involve different methods for extracting relevant information from malware samples. Figure 1 presents the difference between static and dynamic analysis.

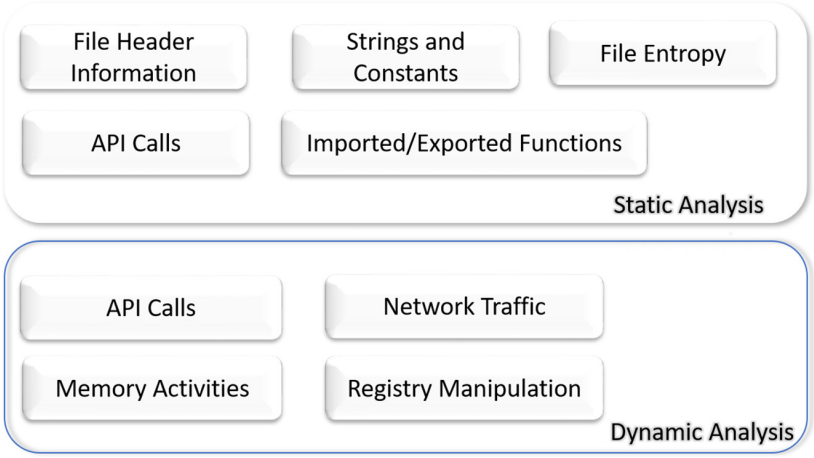

Fig. 1. Static and dynamic analysis

Static analysis involves examining the binary code or the file structure of a malware sample without actually executing it. In static feature selection, relevant attributes are extracted from the malware file itself, typically without running the malware. These attributes include file header information, strings and constants, file entropy, API calls, and import or export functions [18].

Dynamic analysis involves running malware in a controlled environment, like a sandbox, to observe its behavior during execution. Dynamic feature selection focuses on attributes that can be observed only when the malware is running. These attributes might include API calls, memory activities, system calls, registry manipulation, and network traffic [19] [20].

In practice, a combination of both static and dynamic analysis is often used to achieve comprehensive malware analysis. Static analysis provides quick insights into the structure of malware, while dynamic analysis reveals its actual behavior during execution. This combined approach helps security researchers and analysts better understand and classify malware.

In [21], both ML and deep learning techniques were used to detect malware targeting Android devices. The methods were evaluated using two malware datasets, CICAndMal2017 and DREBIN, using different performance metrics such as F-score, accuracy, and recall. The results show that SVM performed best on the CICAndMal2017 dataset, while long short-term memory (LSTM) outperformed on the DREBIN dataset. It is worth noting that the dataset sizes used in this study were relatively small, with only 676 samples for CICAndMal2017 and 15,031 for DREBIN.

In [22], a hybrid ML method for detecting Android malware by integrating LSTM and bidirectional long short-term memory (BLSTM) is introduced. This approach was evaluated using the CICAndMal2017 dataset, focusing on accuracy and FPR. However, the dataset was significantly down sampled, with only 41,233

records compared to the original dataset size. Although the method achieved a high accuracy of 98.7%, it exhibited a high FPR. Moreover, the authors in [23] introduced an Android malware detection system employing a code deobfuscation technique. This study demonstrated that code deobfuscation can effectively recover obscured information. They also identified interaction words based on feature interactions.

In [24], an investigation of the security challenges in IoT networks has been presented, focusing on the use of ML algorithms for anomaly detection in network data. The study assesses the performance of various ML algorithms in comparable scenarios, conducting a comparative analysis based on different parameters and methodologies. Using the IoT-23 dataset for evaluation, the RF algorithm achieved an accuracy of 99.5%.

Utilizing static code analysis with a multi-criteria decision-making (MCDM) approach is proposed in [7]. They used a risk-based fuzzy analytical hierarchy process (AHP) to evaluate Android mobile applications in terms of malware detection. Their method evaluated mobile malware detection techniques based on permission-based characteristics.

In [25], the IoT-23 dataset was used to evaluate their proposal. They focused on achieving high accuracy in malware detection using ML and deep learning models. The results were 99.9% and 99.8% by using the DT model and RF algorithms, respectively.

These studies collectively highlight various approaches to tackling Android malware detection, ranging from ML, and deep learning techniques to code deobfuscation and permission-based analysis. Each approach contributes to the ongoing efforts to enhance mobile device security.

In [26], a novel multi-view deep learning for Android malware detection has been introduced. Their model demonstrated notable performance, achieving weighted average detection rates of 91% and 81%. These results were obtained through testing on the DREBIN dataset.

Another noteworthy contribution comes from [27], who introduced DroidNNet, a deep learning-based approach for malware classification. Droid-NNet is a deep learning framework that outperforms existing state-of-the-art ML methods in malware classification. They evaluated their proposed model using DREBIN-215, a dataset comprising 215 Android apps.

Table 1 provides an overview of various malware detection techniques found in the literature. These techniques differ based on the method used to analyze the malware, which can be either static or dynamic. Additionally, they vary in the type of detection learning employed, which may involve ML or deep learning.

These techniques have been tested on different benchmark datasets. Collectively, these studies showcase the ongoing efforts to develop innovative techniques for Android malware detection, with a focus on deep learning models and diverse benchmark datasets.

Table 1. Comparison between various malware detection techniques

Reference	Method	Technique	Database
[26]	Static	Deep Learning	DREBIN
[29]	Static	SVM	DREBIN
[21]	Dynamic	Deep learning	DREBIN, CICAndMal2017
[22]	Dynamic	Hybrid (LSTM and BLSTM)	CICAndMal2017

(Continued)

Table 1. Comparison between various malware detection techniques *(Continued)*

Reference	Method	Technique	Database
[24]	Static	Several Machine learning techniques	IoT-23
[25]	Static	Deep Learning + Machine Learning	IoT-23
[28]	Static	NLP	FFRI
[30]	Static	Deep Learning	Malimg, Microsoft's BIG 2015, MaleVis, and Malic
[31]	Static	graph neural network	AMGP, DB and AMD
[32]	Static	convolution neural networks	AMD
[33]	Static	OWL	DREBIN
Our Proposal	Static	Fusion System based on 2 different systems	CICAndMal2017 and DREBIN

3 BACKGROUND

This section presents brief descriptions of the three key one-class classifiers used in this paper: the Isolation Forest, OC_SVM, and LOF. These classifiers complement the utilization of the M-IForest.

3.1 Isolation Forest

Isolation Forest, often denoted as iForest, was originally developed by Fei Tony Liu and Zhi-Hua Zhou in 2008. The iForest algorithm represents a prominent technique in the field of data anomaly detection. This approach utilizes binary tree structures as its foundational element. Remarkably, iForest exhibits two noteworthy advantages: it demonstrates a linear time complexity with respect to dataset size and imposes minimal memory requirements. This makes it especially suitable for the analysis of large-scale datasets. At its core, the algorithm efficiently approximates data point density and subsequently identifies instances with exceptionally low-density estimates as potential anomalies within the dataset [34].

The training process of the iForest involves two crucial algorithms: the isolation tree (iTree) building process, which is a fundamental component of the iForest algorithm, designed to efficiently detect anomalies (outliers) in a dataset, and the Forest Aggregation algorithm. The anomaly score for a data point is based on an iTree, and can be calculated using Equation 1.

$$S(X, \psi) = 2\frac{-E(h(x)}{c(\psi)} \quad (1)$$

Where E (h(x)) is the expected path length for the data point 'x' in the tree. It's a measure of how deep in the tree the data point is typically isolated during the tree-building process, and c (n) is a constant that represents the average path length for an unsuccessful search in a binary tree of 'n' data points. It can be calculated using Equation 2.

$$c\ (n) = 2H\ (n - 1) - (2(n - 1)/n) \quad (2)$$

Algorithm 1 presents the pseudocode for the iTree-building process. The idea behind the iTree building process is that anomalies are easier to isolate and require fewer splits in the tree, leading to shorter path lengths from the root to the anomaly. By aggregating the results from multiple iTrees in the forest, the iForest algorithm can effectively identify anomalies as data points with shorter average path lengths in the forest, making it a powerful technique for outlier detection, particularly in high-dimensional datasets. Algorithm 2 illustrates the evaluation phase of the classical iForest for any new instance by employing the path length process.

***Algorithm 1*: iTree(I,h,l)**

Input: Input data I, The current height of the tree h, The height limit l
Output: an iTree

```
    if h ≥ l or |I| ≤ 1 then
 2: return exNodeSize ← |I|
    else
 4: let A a list of attributes in I
    select a random attribute a ∈ A
 6: select a random split point p in the range of max and min values of attribute q in I
    I_l ← filter(I, a < p)
 8: I_r ← filter(I, a ≥ p)
    return inNode Left ← iTree(I_l, h + 1, l)
10: Right ← iTree(Ir, h + 1, l),
    Splitattribute ← a,
12: SplitValue ← p
    end if
```

After building a collection of individual isolation trees (subtrees), the Forest Aggregation algorithm 3 combines them to form the final iForest model. This ensemble approach helps improve the overall anomaly detection performance [35], [36].

In summary, the iForest training process involves randomly building a collection of isolation trees, where each tree aims to isolate anomalies by recursively partitioning the data. These individual trees are then combined through an aggregation process to create an effective anomaly detection model. This approach is particularly efficient and scalable for detecting outliers in high-dimensional datasets.

'auto' configures the threshold as in the original paper.

'float' allows contamination values within the range of [0, 0.5].

The major enhancement in the M-iForest lies in the dynamic configuration of the contamination parameter based on the fitness value. This fitness value is computed using Equation 3, which relies on the outlier values associated with the anomaly scores.

$$FV = \max\left[\alpha * \frac{F}{N} + \beta * E(h(x)) + \delta * \frac{1}{\text{samples}}\right] \tag{3}$$

In this context, N denotes the total number of features in the dataset. F represents the total number of features that have been randomly selected for each tree. Additionally, $\alpha = 0.08$, and β and $\delta = 0.46$. Furthermore, E (h(x)) refers to a harmonic number, with Euler's constant being approximately 2.71.

Algorithm 2: PathLength(z, T, e)

Input: a new instance z, set of iTree T, Length of the current path e; initially set to zero
Output: path length of z

if T is an external node then
return e + c(T.size) c(Ψ) is defined in 2
3: end if
a ← attribute(T.Split)
if z_a < Value(T.split) then
6: return PathLength(x, T.left, e + 1)
else za ≥ Value(T.split)
X_r ← filter(I, a ≥ p)
9: return PathLength(z, T.right, e + 1)
end if

Algorithm 3: iForest(I,t, Ψ)

Input: Input data I, The number of trees t, The subsampling size Ψ
Output: set of iTrees t

1: Initialize Forest
2: let height limit l = ceiling($\log_2$, Ψ)
3: for j = 1 to t do
4: I' ← sample(I, Ψ)
5: new Forest ← old Forest ∪ iTree(I', 0, 1)
6: end for
7: return Forest

3.2 Local Outlier Factor

The Local Outlier Factor (LOF algorithm is particularly effective at identifying outliers in datasets with irregular distributions. It measures how much the local density of a specific data point deviates from the densities of its neighboring points. A data point is more likely to be classified as an outlier if its local density is significantly lower than that of its neighbors [37] [38].

3.3 One class support vector machine (OC-SVM)

The OC-SVM, developed by Schölkopf and Smola, is widely used for unsupervised anomaly detection. Unlike traditional SVMs, which classify multiple classes, OC-SVM focuses on distinguishing data points from the origin in a Reproducing Kernel Hilbert Space (RKHS). It aims to maximize the distance between a hyperplane and the origin, treating the origin as the sole negatively labeled instance [39].

4 THE PROPOSED MALWARE SYSTEM

In this paper, we introduce a novel fusion-based malware detection system that uses two parallel subsystems, as shown in Figure 2. Each subsystem is trained on a specific type of data—one on benign applications and the other on malware. This design addresses the common issue of imbalanced data by ensuring that each subsystem focuses solely on one class of data. Both subsystems employ an ensemble approach called *bagging*, which combines three one-class classifiers: the M-iForest algorithm, LOF, and OC-SVM.

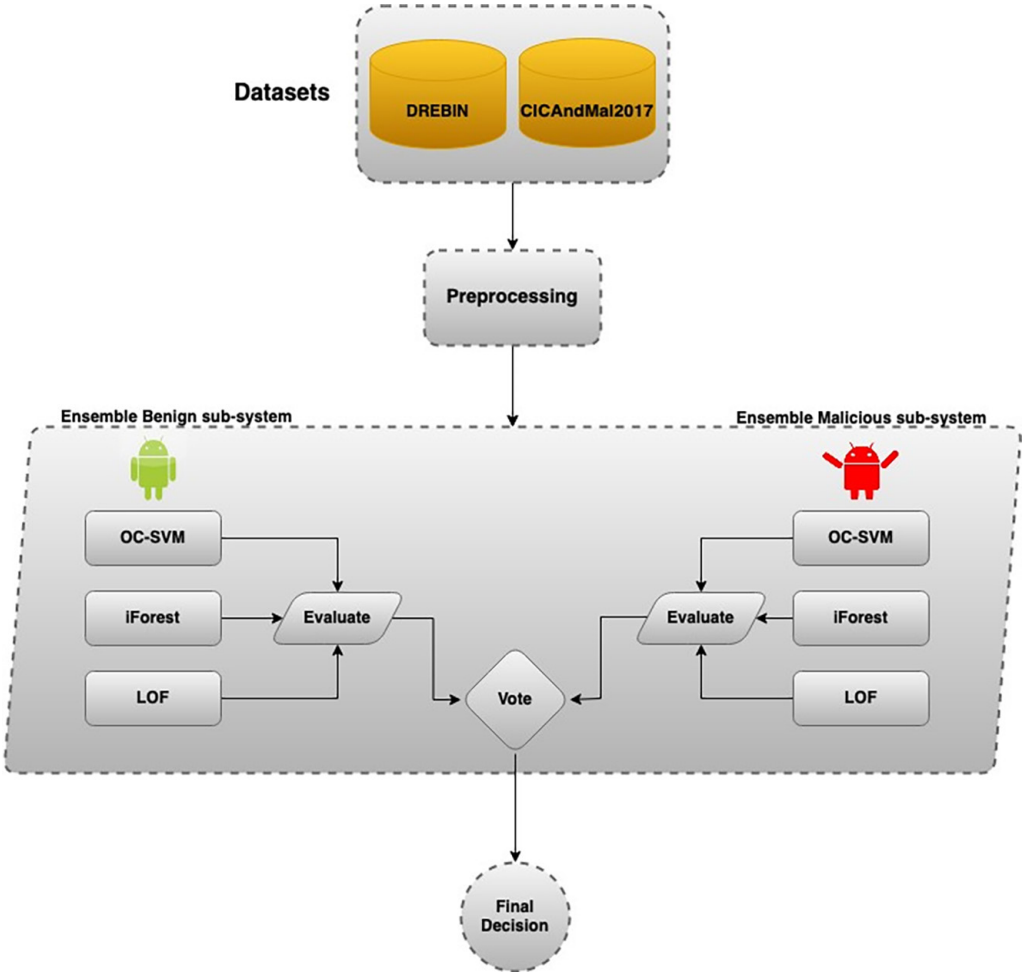

Fig. 2. Malware detection system

The first subsystem, called the *Ensemble Benign Subsystem*, is trained on benign data, while the second, the *Ensemble Malicious Subsystem*, is trained on malware data. By combining the results from both subsystems, the system improves its overall malware detection performance. This dual-subsystem approach is particularly effective because it avoids issues associated with imbalanced data. Each subsystem specializes in its respective data type, which enhances its ability to detect anomalies accurately and reliably. Essentially, this strategy leverages the strengths of both subsystems, making the malware detection process more efficient and trustworthy.

The *Ensemble Benign Subsystem* uses the benign dataset to train its three one-class classifiers, enabling it to make well-informed decisions about what constitutes normal behavior. On the other hand, the *Ensemble Malicious Subsystem* is trained exclusively on malware data, allowing it to focus on identifying malicious patterns. By dedicating each subsystem to its specific data type, the system ensures comprehensive coverage and robust detection capabilities, ultimately improving accuracy and reliability.

Before training, the data undergoes a crucial preprocessing step. This includes normalizing the data, removing duplicate or irrelevant records, and selecting the most important features. The dataset is then split into two groups based on their labels: benign and malicious. This separation ensures that each subsystem is trained on clean, optimized data tailored to its specific task. By focusing on well-prepared datasets, the subsystems can more effectively learn to distinguish between normal

and suspicious patterns. This careful preparation not only boosts the accuracy of anomaly detection but also strengthens the overall reliability of the system.

The final decision-making process of the malware detection system comes from combining the outputs of the two subsystems. The system uses a voting mechanism based on the values shown in Table 2. In this study, a value of *'1'* indicates malware, while *'0'* represents benign instances. Based on this, the system can classify an application as benign, malware, or an anomaly.

Table 2. The fusion system for voting based on the two subsystems

Malicious_System	Benign_System	Final Decision
1	0	Malware
1	1	Malware
0	0	Benign
0	1	Anomaly

In our model, if the first subsystem identifies an application as benign but the second flags it as malware, the final decision is labeled as malware. On the other hand, if both subsystems make classification errors, the application is marked as an anomaly. How these anomalies are handled depends on the system's configuration—they can either be blocked or allowed, depending on what is deemed appropriate.

5 EXPERIMENTS AND DISCUSSION

This section presents the datasets used to evaluate the performance of the proposed system, which are CICAndMal2017 dataset and the DREBIN dataset. Moreover, it provides details about the dataset's specifications, the performance metrics utilized, and the experimental setup parameters.

The CICAndMal2017 dataset and the DREBIN Android dataset are publicly available and free to access, allowing researchers to explore and evaluate their innovative techniques and develop new methodologies. Moreover, they are widely considered benchmark dataset, and they are used by several proposals; this offers a comparison between each other's. In addition, it consists of various malware families, and it has a rich set of features. Additionally, they provide a wide range of real-world Android malware scenarios, covering various types of malwares and a wide spectrum of features.

It is very important to highlight the limitations for these datasets, as they focus on static analysis and static features only, without supporting dynamic behavioral analysis. Moreover, the class imbalance in the dataset can significantly affect a classifier's decision-making process if it has not been reprocessed well, potentially leading to biased classification or less accurate results. However, these limitations do not affect our approach because the proposed system is designed to be anomaly-based, making imbalanced data a non-issue. Furthermore, our approach relies on static analysis rather than dynamic analysis, aligning well with the dataset's characteristics.

5.1 CICAndMal2017 dataset

The CICAndMal2017 dataset encompasses a diverse range of malware categories, including adware, ransomware, SMS malware, and scareware, complemented by an

extensive set of 80 traffic features. To provide context, this dataset comprises 1,700 instances of benign ware and 426 instances of malware, as detailed in [40].

In our research, we leveraged the entire CICAndMal2017 dataset, adopting an 80/20 data split for training and testing, respectively. Notably, the adware and ransomware categories encompass a total of 10 distinct malware families, while scareware and SMS malware feature 11 unique malware families. To facilitate ML processes, we uniformly labeled all malware families as "1" to signify their malicious nature. Conversely, benign data was derived from the benign 2017 version and appropriately labeled with a "0."

5.2 DREBIN dataset

The DREBIN dataset focuses on Android malware and is introduced in [29]. This extensive dataset spans from August 2010 to October 2012 and encompasses a staggering 131,611 programs, covering both benign and malicious software. Within this extensive collection, there are 12,000 benign applications and 5,560 instances of malware. The dataset comprises samples from various sources, including popular app stores, international markets, and additional platforms such as Android-specific websites, malware discussion forums, and cybersecurity blogs. It also incorporates data from the Android Malware Genome Project, enhancing its diversity and complexity [41].

Building a dataset from an Android application involves extracting features from two key sources: the Dex code and the Android manifest. The Android manifest file, usually named 'androidmanifest.xml,' contains crucial information about the application. Simultaneously, the Dex code, which is used across Google and Android Linux-based mobile platforms, plays a significant role in feature extraction. Table 3 offers a detailed summary of the feature sets derived from both the manifest and the Dex code, providing essential data for further analysis [42].

Table 3. Features of dex code and manifest files

Features	Description
Manifest File	
Hardware Components	Contains essential hardware requests, such as access to the smartphone's camera, touchscreen, or GPS for location identification
Filtered Intents	Intents in Android enable communication both within inter-process and between intra-process. Acting as a passive data structure, intents serve as synchronous messages that allow different components and applications to share information. This mechanism is essential for the seamless integration and functionality of diverse application components within the Android ecosystem.
Requested permissions	Intents in Android enable communication both within a single process and between multiple processes. Acting as a passive data structure, intents serve as synchronous messages that allow different components and applications to share information. One of the key security features platform in Android is involving the use of requested permissions
App Components	Application components fall into four categories: services, activities, broadcast receivers, and content providers. Developers can declare each type of component in the manifest file.

(Continued)

Table 3. Features of dex code and manifest files *(Continued)*

Features	Description
Disassembled Code	
Used permission	The permissions outlined in the manifest and those actually used, as identified through code disassembly.
Network Addresses	Any hostname, URL or IP address, found in the disassembled code is included in this category. This ensures that all potential network indicators are captured for further analysis
Suspicious API calls	Malware applications frequently utilize API calls to obtain private information or resources. These API calls can be categorized into four types: SMS message transmission and reception, access to sensitive data, online communication, and obfuscation.
Restricted API Calls	Several essential API calls are restricted access. This mechanism identifies API calls within the Dex code that lack the necessary permissions.

DREBIN dataset structure. The dataset is comprehensive, encompassing the entire source code of malicious or malware applications, each of which is uniquely identified by its SHA256 signature. This extensive dataset comprises numerous files, with the feature vector folder containing files for every application, all of which are named after their respective SHA256 signatures and contain the specific feature vectors associated with each application.

In addition to the feature vector files, there is a family label file that consolidates all the application signatures, each paired with its corresponding label. These labels categorically denote whether an application belongs to one of the 179 malware families present in the dataset or is benign in nature, as documented in [29].

To facilitate model training and evaluation, the dataset further includes a dataset splits folder, housing a total of 10 distinct subsets. Each subset is designed to comprise subsets for training, testing, and validation. The file structure within these subsets adheres to the previously mentioned format, maintaining consistency with the feature vectors.

5.3 Performance metrics

This subsection outlines the essential performance metrics for assessing the various approaches discussed in this study, along with their calculation formulas as referenced in [43]:

- False positive rate (FPR): The FPR, as depicted in Equation (4), quantifies the percentage of normal-class instances that are incorrectly classified as attackers.

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

- Sensitivity (True positive rate – TPR): The TPR, outlined in Equation (5), measures the proportion of actual attacks that are accurately identified by the model.

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

- Accuracy: Accuracy, described by Equation (6), signifies the ratio of correctly classified instances to the total possible categories.

$$Accuracy = \frac{TN + TP}{TN + FN + TP + FP} \tag{6}$$

- F-score (F-measure): The F-score, expressed in Equation (7), offers a comprehensive assessment of the model's accuracy by considering both precision and recall values.

$$F-Score = \frac{2TP}{2TP + FP + FN} \tag{7}$$

- Precision (Positive predictive value): Precision, as given in Equation (8), quantifies the accuracy of positive predictions made by the model.

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

5.4 Experimental setup

In this section, we provide a comprehensive overview of the experimental environment and the initialization parameters employed in our study. The experiments were performed on a computing system featuring an Intel Core i7 processor, 16 GB of RAM, and running a 64-bit Windows 10 operating system. Furthermore, we utilized the Anaconda Python framework, specifically version 5.1, as the foundation for implementing our anomaly malware detection system.

To ensure transparency and reproducibility, we present the initial values of the parameters utilized in our proposed model. The experimental setup for M-iForest parameters was conducted over 25 runs. Moreover, the max-samples parameter was set to 300, the number of n-estimators was set to 100, and contamination as defined in Equation 3 with the values for β, α, and δ were 0.46, 0.08, and 0.46, respectively.

This information serves as a foundational reference for understanding the configuration and setup of our experiments, facilitating the replication of our research findings.

6 RESULTS AND DISCUSSION

This paper evaluates the performance of our proposed anomaly malware detection fusion system using two datasets, namely DREBIN and CICAndMal2017.

6.1 DREBIN results

Various scenarios were evaluated for each subset in the DREBIN dataset. Tables 4 and 5 offer a comprehensive comparison of different one-class classifiers based on models trained with benign data. Table 4 highlights the performance of traditional isolation forests and their modified versions, assessing accuracy, F-score, and FPR across 10 subsets of the DREBIN dataset. The findings demonstrate that the M-iForest significantly improves the detection of malware anomalies. Specifically, in terms of accuracy, F-score, and FPR, the classic outperforms both OC-SVM and LOF. Table 5 presents the experimental results for LOF and OC-SVM. It can be noted that LOF achieves a lower FPR compared to OC-SVM. However, in terms of F-score and accuracy, the results are nearly identical, with only minor variations.

Table 4. Comparison between Modified Isolation Forest and Classic Isolation Forest for DREBIN dataset based on different subsets

Subsets	Modified Isolation Forest			Classic Isolation Forest		
	F-score	Accuracy	FPR	F-score	Accuracy	FPR
Subset I	0.963	0.991	0.222	0.860	0.926	0.360
Subset II	0.969	0.997	0.205	0.842	0.930	0.342
Subset III	0.953	0.985	0.237	0.843	0.915	0.375
Subset IV	0.945	0.977	0.272	0.835	0.908	0.390
Subset V	0.969	0.997	0.193	0.875	0.939	0.338
Subset VI	0.957	0.986	0.215	0.864	0.927	0.359
Subset VII	0.962	0.985	0.253	0.843	0.926	0.363
Subset VII	0.958	0.973	0.274	0.832	0.905	0.398
Subset IX	0.946	0.976	0.274	0.836	0.909	0.405
Subset X	0.959	0.985	0.228	0.863	0.928	0.362
Average	**0.9581**	**0.9852**	**0.2373**	**0.8493**	**0.9213**	**0.369**

It can be noted that the M-iForest achieves a significant improvement, with approximately a 6.93% increase in accuracy and a 12.81% boost in F-score. This improvement indicates the enhanced capability of the M-iForest in detecting anomalies within different datasets. The increased accuracy suggests that the model is better at correctly identifying malware, while the higher F-score indicates a better balance between precision and recall. These enhancements demonstrate the potential of the M-iForest to significantly improve the performance of the overall subsystems, which directly enhances the fusion system's ability to detect anomalies.

Table 5. Comparison results between OC-SVM and LOF using DREBIN subsets

Subsets	OC-SVM			LOF		
	F-score	Accuracy	FPR	F-score	Accuracy	FPR
Subset I	0.751	0.822	0.470	0.741	0.862	0.175
Subset II	0.742	0.853	0.481	0.742	0.840	0.183
Subset III	0.766	0.859	0.448	0.753	0.862	0.179
Subset IV	0.766	0.848	0.465	0.757	0.863	0.174
Subset V	0.757	0.838	0.460	0.776	0.875	0.191
Subset VI	0.762	0.850	0.438	0.776	0.864	0.176
Subset VII	0.762	0.876	0.432	0.768	0.868	0.195
Subset VII	0.759	0.864	0.450	0.773	0.862	0.183
Subset IX	0.736	0.849	0.464	0.756	0.845	0.178
Subset X	0.758	0.857	0.467	0.765	0.832	0.182
Average	**0.7559**	**0.8516**	**0.4582**	**0.7614**	**0.8580**	**0.1819**

Table 6 presents a comparison between our proposed system and several malware detections using the DREBIN dataset. The evaluation is based on metrics

such as FPR, accuracy, and F-score. Our approach outperforms some other related malware detection methods.

Notably, when compared to the approach in [6], our system achieves higher F-score and accuracy, with values of 95.67%, and 98.7%, respectively. In [6], the authors apply several ML algorithms, with RF showing the best results. In [7], deep learning is used, achieving an accuracy of 90.54, while in [14], the RF is used to detect the malware, achieving an accuracy of 97.47.

In [44], J48 DT algorithm was used. The results showed an accuracy value of 98.4% and a FPR of 0.80. Moreover, in [45], MalDozer was proposed, where the approach classifies malware using deep learning techniques. Additionally, the data was split into several folds, and the best results were achieved when 10-fold was used. In [46], feature selection was employed to choose the best features for the DREBIN dataset, and feedforward deep neural network models were used as classifiers. The findings indicate an improvement in accuracy of approximately 0.06 and 0.01 in FPR when compared to our method. This improvement can be attributed to the fact that our proposed fusion system does not utilize feature selection.

Table 6. Comparison results between several malware detection systems using DREBIN dataset in the literature

Ref.	FPR	F-Score%	Accuracy%
[6]	–	94.0%	94.33%
[7]	–	–	90.54%
[14]	–	–	97.47
[44]	0.80	–	98.4%
[45]	0.45	–	99.2%
[46]	0.22	–	98.76%
The proposed system	**0.23**	**95.67%**	**98.70%**

6.2 CICAndMal2017 results

Table 7 provides a comprehensive evaluation of our proposed ensemble approach, which employ three one-class classifiers, within the context of the first subsystem dedicated to malicious activity detection. This evaluation encompasses both the CICAndMal2017 and DREBIN datasets, focusing on critical performance metrics such as TPR, FPR, accuracy, F-score, and area under the curve (AUC).

Table 7. Analyzing the performance of the malicious detection sub-system with different datasets

Datasets	Approach	FPR	TPR	F-score	Accuracy	AUC
DREBIN	OC-SVM	0.4986	0.8162	0.7389	0.8304	0.7020
	M-iForest	0.2489	0.9456	0.9197	0.9598	0.9478
	LOF	0.2150	0.7381	0.7412	0.8276	0.7891
	Malware Sub-System	**0.276**	**0.9351**	**0.9141**	**0.9235**	**0.9135**
CICAndMal2017	OC-SVM	0.0310	0.9241	0.8952	0.9132	0.9232
	M-iForest	0.0758	0.9791	0.9398	0.9836	0.9589
	LOF	0.1530	0.8365	0.8120	0.8751	0.8677
	Malware Sub-System	**0.0942**	**0.9723**	**0.9534**	**0.9714**	**0.9681**

Notably, the M-IForest consistently outperforms LoF and OCSVM for both datasets. Furthermore, the ensemble malicious subsystem approach exhibits remarkable performance improvements, achieving an accuracy of 97.14% and 92.35%, as well as an F-score of 95.34% and 91.41% for CICAndMal2017 and DREBIN, respectively.

In Table 8, we shift our focus to the second subsystem, which is trained on benign datasets for both CICAndMal2017 and DREBIN. Interestingly, the ensemble benign subsystem consistently outperforms the malicious subsystem, and M-IForest exhibits superior performance compared to OC-SVM and local outlier factor.

Table 8. Analyzing the performance of the benign sub-system with different datasets

Dataset	System	FPR	TPR	Accuracy	AUC	F-score
CICAndMal2017	M-iForest	0.0553	0.9893	0.9873	0.9692	0.9690
	LoF	0.1320	0.8509	0.8965	0.8954	0.8489
	OC-SVM	0.0240	0.9651	0.9369	0.9456	0.9065
	Benign Sub-System	**0.0325**	**0.9876**	**0.9924**	**0.9856**	**0.9782**
DREBIN	M-iForest	0.2321	0.9698	0.9821	0.9768	0.9567
	LOF	0.1820	0.7581	0.8576	0.8091	0.7612
	OC-SVM	0.4586	0.8462	0.8504	0.7320	0.7589
	Benign Sub-System	**0.3572**	**0.9623**	**0.9534**	**0.9368**	**0.9467**

Table 9 presents the results of both ensemble subsystems and the fusion system, examining TPR, FPR, accuracy, F-score, and AUC. It's worth noting that the benign subsystem consistently outperforms the malicious subsystem. However, the proposed fusion system leverages the strengths of both subsystems, enhancing the detection performance of malicious activities.

Table 9. Performance evaluation of the fusion system using different datasets

Dataset	System	FPR	TPR	Accuracy	AUC	F-score
CICAndMal2017	Benign sub-system	0.0325	0.9876	0.9924	0.9856	0.9782
	Malware sub-system	0.0942	0.9723	0.9714	0.9681	0.9534
	Fusion System	**0.0467**	**0.9885**	**0.9901**	**0.9836**	**0.9658**
DREBIN	Malware sub-system	**0.2758**	**0.9351**	**0.9235**	**0.9135**	**0.9141**
	Benign sub-system	0.3572	0.9623	0.9534	0.9368	0.9467
	Fusion System	**0.1895**	**0.9704**	**0.9705**	**0.9387**	**0.9587**

Table 10 presents a detailed comparison among various related proposals using the CICAndMal2017 dataset. Notably, our proposal achieves better results compared to others in terms of precision, recall, and accuracy. However, it is important to note that not all previous proposals report all performance metrics as [14], [47], [48], and [49].

In [14], SVM is used to classify malware and benign classes, but it does not address the class imbalance in the CICAndMal2017 dataset or preprocessing the data. As a result, it achieves a relatively low accuracy of 73.22%. In [47], experiments are

conducted based on two scenarios. In the first scenario, the classifier is trained on a single dataset containing various types of ransomwares. In the second scenario, it is trained on datasets from 10 different ransomware families. The results show that the RF method outperforms in both cases.

In [48], the proposed method consists of several phases to evaluate the CICAndMal2017 dataset. The first phase involves analysis and feature extraction, followed by data cleaning. In the second phase, feature selection is performed using three classifiers: LightGBM, RF, and recursive feature elimination (RFE). A voting technique is employed to determine the final decision. Using this ensemble technique, the malware detection accuracy achieved was 87.75%. Moreover, the RF achieves the best results compared to the other classifiers: 88.3 and 85.8 for recall and precision, respectively.

In [50], they achieved better recall compared to our proposal, but our proposal outperforms theirs in terms of precision and accuracy.

In [52], MDADroid is proposed based on three phases: preprocessing, mapping construction—which consists of heterogeneity construction and functionality API mapping–and Android malware detection. It can be noted that their approach achieves high results in terms of accuracy and precision compared to recall, which was 0.8517.

The proposed fusion system for malware detection outperforms previous approaches. It consists of two subsystems trained using anomaly detection and employs an ensemble approach to enhance the overall model's performance.

Table 10. Comparison results between several malware detection systems in the literature using CICAndMal2017 dataset

Ref.	Method	Recall	Precision	Accuracy
[14]	SVM	–	–	73.22
[47]	API-Calls + Network Flows	95.30	95.30	–
[48]	RF	–	–	82.80
[49]	RF	88.3	85.8	–
[50]	CNN-LSTM	98.19	97.29	97.29
[52]	MDADroid(stacking)	0.8517	0.9507	0.9614
The Proposed Model	**Fusion System**	**95.85**	**97.89**	**99.01**

6.3 Statistical analysis

Statistical analysis is crucial for evaluating and assessing the importance of differences between different approaches in this section, we thoroughly analyze the experimental results to perform a statistical comparison between the M-iForest model and the original iForest model proposed by [31].

First, the Shapiro-Wilk test was used with a significance level (alpha) of 0.05 to assess the normality of the results' distribution, as presented in Table 12. To ensure fairness, a random sample of 20 findings from our approach was selected to determine the normal distribution of our results.

Second, a T-test is employed to calculate the p-value of the test statistics, allowing us to determine the statistical significance of the observed sample data. The p-value is a critical metric, referring to the likelihood of a specific event occurring. In this

analysis, we set the significance threshold (alpha) at 0.05, meaning that the observed results become more statistically significant as the p-value decreases [51].

During the statistical analysis, results from five DREBIN subsets were examined for both the M-iForest and iForest models.

Table 11 provides a summary of the Mean, Standard Error (Std.Err), Standard Deviation (Std.Dev), and P-Value for accuracy and F-score. It is noteworthy that the significance level (sig) is below the critical threshold of 0.05. This outcome indicates that the results obtained from the M-iForest version indeed exhibit statistical significance.

In summary, the statistical analysis confirms the statistical significance of the M-iForest model's performance compared to the original iForest model when applied to the DREBIN dataset under the specified experimental conditions.

Table 11. Statistical analysis for DREBIN dataset

Subsets	Metric	Modified Isolation Forest			Classic Isolation Forest			P-Value
		Mean	Std.Dev	Std.Err	Mean	Std.Dev	Std.Err	
Subset II	F-Score	95.1301	0.5455	0.2198	87.1643	1.74920	0.3870	0.001
Subset II	Accuracy	98.1923	0.6929	0.2086	91.9187	1.61385	0.4720	0.001
Subset IV	F-Score	97.4560	0.9717	0.1274	85.5325	0.7417	0.3543	0.001
Subset IV	Accuracy	99.4560	0.7972	0.3250	92.5325	1.3332	0.7983	0.001
Subset VI	F-Score	92.8960	1.3452	0.3282	83.3252	1.3017	0.6040	0.001
Subset VI	Accuracy	96.5335	1.1496	0.3245	80.9192	2.7862	0.5137	0.001
Subset VII	F-Score	93.7960	0.6932	0.2338	84.5327	1.6147	0.4593	0.001
Subset VII	Accuracy	98.5235	0.7966	0.2730	90.7160	2.2736	0.3978	0.001
Subset X	F-Score	95.7560	1.216	0.3512	88.2355	2.3657	0.5235	0.001
Subset X	Accuracy	99.7635	1.2651	0.3060	92.9189	1.1946	0.6128	0.001

6.4 Comparative analysis

This section provides a detailed comparison of the tools that are used in malware detection. Table 12 summarizes the advantages and limitations of previous related tools and approaches designed for malware detection. It can be observed that deep learning generally offers better performance and can analyze dynamic behavior. However, its effectiveness comes with limitations, such as high computational time based on its architecture. Whereas it takes a huge time, so it can have an issue when scaling up. Also, it requires vast amounts of labeled data for effective training and learning, as [7], [45], [46], and [50].

When using traditional ML algorithms for malware detection, the RF classifier is considered the most powerful and robust algorithm with excellent accuracy, making it suitable for a wide range of tasks, as mentioned in [14], [48], and [49]. The limitations for the RF, it needs computational cost and it requires careful tuning to maximize its potential. Employing an ensemble approach will enhance the performance of the model, but it can be complex to construct several classifiers and link them with each other, as in [50].

Table 12. Summarized for several approaches

Ref	Tool	Advantages	Limitations
[7]	Deep learning	More accurate and reliable	Not scalable
[14]	Random forest	Good performance	High computation cost
[45]	Deep learning	High accuracy	High computation cost
[46]	Deep learning + feature selection	Handle large volumes of data	High computation cost
[48]	Random forest	Good performance	High computation cost
[49]	Random forest	Good performance	High computation cost
[50]	Deep learning	High performance	High computation cost
[52]	Stacking method	Improve the prediction and reduce the overfitting	The complexity based on the models that are used
Our Approach	**Fusion system**	**Improve the performance, handle the imbalanced data**	**Based on the complexity of the classifiers**

7 CONCLUSIONS

The vast evolution of malware highlights the need for innovative and intelligent approaches to address emerging threats. In this paper, we introduce a new approach for anomaly malware detection based on two subsystems that work in parallel, with each subsystem being carefully trained on a specific type of data, either malicious or benign. This approach helps prevent the issues associated with imbalanced datasets. Additionally, each subsystem is based on an ensemble approach that leverages an enhanced version of the unsupervised learning algorithm. The ensemble consists of OC-SVM, LOF, and M-iForest. This innovative choice of classifier empowers our system to effectively detect anomalous malware patterns. Using two subsystems offers the advantage of allowing each subsystem to specialize in a specific type of learning, resulting in improved performance compared to training on diverse types of labeled data. Moreover, the ensemble approach involves multiple classifiers analyzing the same data and reaching a final decision based on majority voting. This method adds an additional layer of accuracy to the overall system. The evaluations were conducted for each subsystem and the fusion system using the DREBIN and CICAndMal2017 datasets. The experimental results demonstrated impressive performance metrics on the DREBIN dataset, with our model achieving an accuracy of 97.05% and an F-score of 95.87%. Additionally, when evaluated on the CICAndMal2017 dataset, our system achieved outstanding accuracy and an F-score of 99.01% and 96.58%, respectively. The comparative analysis with other malware approaches that utilized the DREBIN and CICAndMal2017 datasets clearly demonstrates the superiority of our system in terms of accuracy, F-score, FPR, and TPR. This highlights the effectiveness of our approach in addressing malware detection challenges and emerging threats. As future work, we plan to develop a comprehensive and enriched Android dataset to facilitate the discovery of new malware features. Furthermore, the behavioral patterns of next-generation malware will be analyzed, and a new approach will be designed for selecting appropriate ML algorithms to further enhance detection performance.

8 REFERENCES

[1] E. Barrionuevo and W. Ticona, "Integration of mobile and web applications to prevent crime," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 19, no. 2, pp. 111–125, 2025. https://doi.org/10.3991/ijim.v19i02.52839

[2] G. Mwakajwanga and O. Mwambe, "Digital-signature oriented steganography approach against man-in-the-middle attack," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 18, no. 16, pp. 158–173, 2024. https://doi.org/10.3991/ijim.v18i16.48709

[3] M. Hakiki, F. Halomoan, R. Hidayah, Y. Zunarti, and V. Y. Yanti, "CT-mobile: Enhancing computational thinking via android graphic design app," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 18, no. 13, pp. 4–19, 2024. https://doi.org/10.3991/ijim.v18i13.47711

[4] O. Abualghanam, M. Qatawneh, and W. Almobaideen, "A survey of key distribution in the context of internet of things," *Journal of Theoretical and Applied Information Technology*, vol. 97, no. 22, pp. 3217–3241, 2019.

[5] O. AbuAlghanam, L. Albdour, and O. Adwan, "Multimodal biometric fusion online handwritten signature verification using neural network and support vector machine," *International Journal of Innovative Computing, Information and Control*, vol. 17, no. 5, pp. 1691–1703, 2021. https://doi.org/10.24507/ijicic.17.05.1691

[6] M. S. Rana, C. Gudla, and A. H. Sung, "Evaluating machine learning models for Android malware detection: A comparison study," in *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, 2018, pp. 17–21. https://doi.org/10.1145/3301326.3301390

[7] J. M. Arif, M. F. Ab Razak, S. R. T. Mat, S. Awang, N. S. N. Ismail, and A. Firdaus, "Android mobile malware detection using fuzzy AHP," *Journal of Information Security and Applications*, vol. 61, p. 102929, 2021. https://doi.org/10.1016/j.jisa.2021.102929

[8] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and API calls tracing," in *2012 Seventh Asia Joint Conference on Information Security*, 2012, pp. 62–69. https://doi.org/10.1109/AsiaJCIS.2012.18

[9] V. Kouliaridis and G. Kambourakis, "A comprehensive survey on machine learning techniques for Android malware detection," *Information*, vol. 12, no. 5, p. 185, 2021. https://doi.org/10.3390/info12050185

[10] K. Dillon, "Feature-level malware obfuscation in deep learning," *arXiv preprint arXiv.2002.05517*, 2020. https://doi.org/10.48550/arXiv.2002.05517

[11] T. Ban, T. Takahashi, S. Guo, D. Inoue, and K. Nakao, "Integration of multi-modal features for Android malware detection using linear SVM," in *2016 11th Asia Joint Conference on Information Security (AsiaJCIS)*, 2016, pp. 141–146. https://doi.org/10.1109/AsiaJCIS.2016.29

[12] M. Masdari and H. Khezri, "A survey and taxonomy of the fuzzy signature-based intrusion detection systems," *Applied Soft Computing*, vol. 92, p. 106301, 2020. https://doi.org/10.1016/j.asoc.2020.106301

[13] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018. https://doi.org/10.1109/TII.2017.2789219

[14] A. Al Sharah, Y. Abu Alrub, H. Abu Owida, E. Abu Elsoud, N. Alshdaifat, and H. Khtatnaha, "An adaptive framework for classification and detection of android malware," *International Journal of Interactive Mobile Technologies*, vol. 18, no. 21, pp. 59–73, 2024. https://doi.org/10.3991/ijim.v18i21.49669

[15] C. Palma, F. Artur, and F. Mário, "Explainable machine learning for malware detection on android applications," *Information*, vol. 15, no. 1, p. 25, 2024. https://doi.org/10.3390/info15010025

[16] S. Navaneethan and S. Udhaya Kumar, "ScanSavant: Malware detection for android applications with explainable AI," *International Journal of Interactive Mobile Technologies,* vol. 18, no. 19, pp. 171–181, 2024. https://doi.org/10.3991/ijim.v18i19.49437

[17] J. Abhishek *et al.*, "Malware analysis using transformer based models: An empirical study," in *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, 2024, pp. 858–865. https://doi.org/10.5220/0012855100003767

[18] H. Wu, N. Luktarhan, G. Tian, and Y. Song, "An Android malware detection approach to enhance node feature differences in a function call graph based on GCNS," *Sensors*, vol. 23, no. 10, p. 4729, 2023. https://doi.org/10.3390/s23104729

[19] A. Muzaffar, H. R. Hassen, H. Zantout, and M. A. Lones, "Droiddissector: A static and dynamic analysis tool for android malware detection," *Advance is Big Data and Cloud Computing*, vol. 645, pp. 147–155, 2018. https://doi.org/10.1007/978-981-10-7200-0_13

[20] J. A. Herrera-Silva and M. Hernández-Álvarez, "Dynamic feature dataset for ransomware detection using machine learning algorithms," *Sensors*, vol. 23, no. 3, p. 1053, 2023. https://doi.org/10.3390/s23031053

[21] H. Alkahtani and T. H. Aldhyani, "Artificial intelligence algorithms for malware detection in Android-operated mobile devices," *Sensors*, vol. 22, no. 6, p. 2268, 2022. https://doi.org/10.3390/s22062268

[22] M. Ahmad, D. Javeed, M. Shoaib, N. Younas, and A. Zaman, "An efficient approach of deep learning for Android malware detection," *United International Journal for Research & Technology*, vol. 2, no. 11, p. 2586, 2021.

[23] Y.-C. Chen, H.-Y. Chen, T. Takahashi, B. Sun, and T.-N. Lin, "Impact of code deobfuscation and feature interaction in android malware detection," *IEEE Access*, vol. 9, pp. 123208–123219, 2021. https://doi.org/10.1109/ACCESS.2021.3110408

[24] S. Nicolas-Alin, "Machine learning for anomaly detection in IoT networks: Malware analysis on the IoT-23 data set," Bachelor's thesis, University of Twente, (2020). https://purl.utwente.nl/essays/81979

[25] G. Devika Rajiv, "Effective approach for malware detection using machine learning and deep learning for IoT-devices," Thesis, Dublin, National College of Ireland, 2023. [Online]. Available: https://norma.ncirl.ie/id/eprint/6912

[26] S. Millar, N. McLaughlin, J. M. del Rincon, and P. Miller, "Multi-view deep learning for zero-day android malware detection," *Journal of Information Security and Applications*, vol. 58, p. 102718, 2021. https://doi.org/10.1016/j.jisa.2020.102718

[27] M. Masum and H. Shahriar, "Droid-NNet: Deep learning neural network for android malware detection," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 5789–5793. https://doi.org/10.1109/BigData47090.2019.9006053

[28] M. Mimura and R. Ito, "Applying NLP techniques to malware detection in a practical environment," *International Journal of Information Security*, vol. 21, pp. 279–291, 2022. https://doi.org/10.1007/s10207-021-00553-8

[29] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of Android malware in your pocket," *NDSS*, 2014. https://doi.org/10.14722/ndss.2014.23247

[30] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient densenet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, 2021. https://doi.org/10.3390/e23030344

[31] H. Gao, S. Cheng, and W. Zhang, "Gdroid: Android malware detection and classification with graph convolutional network," *Computers & Security*, vol. 106, p. 102264, 2021 https://doi.org/10.1016/j.cose.2021.102264

[32] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, "Mapas: A practical deep learning-based Android malware detection system," *International Journal of Information Security,* vol. 21, pp. 725–738, 2022. https://doi.org/10.1007/s10207-022-00579-6

[33] H. Alazzam, A. Al-Adwan, O. Abualghanam, E. Alhenawi, and A. Alsmady, "An improved binary owl feature selection in the context of Android malware detection," *Computers*, vol. 11, no. 12, p. 173, 2022. https://doi.org/10.3390/computers11120173

[34] M. Carletti, M. Terzi, and G. A. Susto, "Interpretable anomaly detection with DIFFI: Depth-based isolation forest feature importance," *arXiv preprint arXiv.2007.11117*, 2020. https://doi.org/10.48550/arXiv.2007.11117

[35] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422. https://doi.org/10.1109/ICDM.2008.17

[36] O. AbuAlghanam, H. Alazzam, E. Alhenawi, M. Qatawneh, and O. Adwan, "Fusion-based anomaly detection system using modified isolation forest for Internet of Things," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, pp. 131–145, 2022. https://doi.org/10.1007/s12652-022-04393-9

[37] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000, pp. 93–104. https://doi.org/10.1145/342009.335388

[38] Z. Cheng, C. Zou, and J. Dong, "Outlier detection using isolation forest and local outlier factor," in *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, 2019, pp. 161–168. https://doi.org/10.1145/3338840.3355641

[39] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT press, 2002. https://doi.org/10.7551/mitpress/4175.001.0001

[40] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark Android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–7. https://doi.org/10.1109/CCST.2018.8585560

[41] A. Daniel, S. Michael, G. Hugo, and R. Konrad, "Drebin: Efficient and explainable detection of Android malware in your pocket," in *Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS)*, 2014, pp. 23–36.

[42] S. Michael, E. Florian, S. Thomas, C. F. Felix, and J. Hoffmann, "Mobile-sandbox: Looking deeper into Android applications," in *Proceedings of the 28th International ACM Symposium on Applied Computing (SAC)*, 2013, pp. 1808–1815. https://doi.org/10.1145/2480362.2480701

[43] I. Sohn, "Deep belief network based intrusion detection techniques: A survey," *Expert Systems with Applications*, vol. 167, p. 114170, 2021. https://doi.org/10.1016/j.eswa.2020.114170

[44] A. Zulkifli, I. R. A. Hamid, W. M. Shah, and Z. Abdullah, "Android malware detection based on network traffic using decision tree algorithm," in *International Conference on Soft Computing and Data Mining*, 2018, pp. 485–494. https://doi.org/10.1007/978-3-319-72550-5_46

[45] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for Android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018. https://doi.org/10.1016/j.diin.2018.01.007

[46] S. Selvaganapathy and S. Sadasivam, "Anti-malware engines under adversarial attacks," *International Journal of Computers and Applications*, vol. 44, no. 8, pp. 791–804, 2021. https://doi.org/10.1080/1206212X.2021.1940744

[47] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible Android malware detection and family classification using network-flows and API-calls," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8. https://doi.org/10.1109/CCST.2019.8888430

[48] F. Noorbehbahani, F. Rasouli, and M. Saberi, "Analysis of machine learning techniques for ransomware detection," in *2019 16th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, 2019, pp. 128–133. https://doi.org/10.1109/ISCISC48546.2019.8985139

[49] M. Abuthawabeh and K. W. Mahmoud, "Enhanced Android malware detection and family classification, using conversation-level network traffic features," *Int. Arab J. Inf. Technol.*, vol. 17, no. 4A, 2020. https://doi.org/10.34028/iajit/17/4A/4

[50] G. Mahshid, S. Hashemi, and L. Abdi, "Android malware detection and classification based on network traffic using deep learning," in *2021 7th International Conference on Web Research (ICWR)*, 2021, pp. 71–77. https://doi.org/10.1109/ICWR51868.2021.9443025

[51] W. R. Rice, "Analyzing tables of statistical tests," *Evolution*, vol. 43, no. 1, pp. 223–225, 1989. https://doi.org/10.2307/2409177

[52] J. Yang, H. Li, L. He, T. Xiang, and Y. Jin, "MDADroid: A novel malware detection method by constructing functionality-API mapping," *Computers & Security*, vol. 146, p. 104061, 2024. https://doi.org/10.1016/j.cose.2024.104061

9 AUTHORS

Orieb AbuAlghanam is with the Department of Computer Science, The University of Jordan, Amman, Jordan (E-mail: o.abualganam@ju.edu.jo).

Hadeel Alazzam is with the Department of Information Technology, Yarmouk University, Irbid, Jordan (E-mail: hadeel.alazzam@yu.edu.jo).

Mohammad Qatawneh Department of Networks and Cybersecurity, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan; Department of Information Technology, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan (E-mail: m.qatawneh@ammanu.edu.jo).

Moutaz Alazab is with the Department of Intelligence Systems, Al-Balqa Applied University, Al-Salt, Jordan (E-mail: m.alazab@bau.edu.jo).

Mohammad Al Sharaiah is with the Department of Information Technology, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan (E-mail: m.alsharaiah@ju.edu.jo).