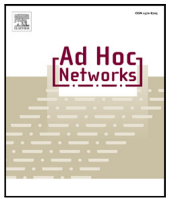
Ad Hoc Networks 161 (2024) 103523

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc



AI-powered malware detection with Differential Privacy for zero trust security in Internet of Things networks

Faria Nawshin [a,b], Devrim Unal [a,*], Mohammad Hammoudeh [c], Ponnuthurai N. Suganthan [a]

[a] KINDI Computing Research Center, College of Engineering, Qatar University, Doha, Qatar
[b] Department of Computer Science & Engineering, College of Engineering, Qatar University, Doha, Qatar
[c] Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

ARTICLE INFO

Keywords:
Privacy-preserving machine learning
Zero trust
Android malware detection
Malware category classification
Differential Privacy
Privacy budget

ABSTRACT

The widespread usage of Android-powered devices in the Internet of Things (IoT) makes them susceptible to evolving cybersecurity threats. Most healthcare devices in IoT networks, such as smart watches, smart thermometers, biosensors, and more, are powered by the Android operating system, where preserving the privacy of user-sensitive data is of utmost importance. Detecting Android malware is thus vital for protecting sensitive information and ensuring the reliability of IoT networks. This article focuses on AI-enabled Android malware detection for improving zero trust security in IoT networks, which requires Android applications to be verified and authenticated before providing access to network resources. The zero trust security model requires strict identity verification for every entity trying to access resources on a private network, regardless of whether they are inside or outside the network perimeter. Our proposed solution, DP-RFECV-FNN, an innovative approach to Android malware detection that employs Differential Privacy (DP) within a Feedforward Neural Network (FNN) designed for IoT networks under the zero trust model. By integrating DP, we ensure the confidentiality of data during the detection process, setting a new standard for privacy in cybersecurity solutions. By combining the strengths of DP and zero trust security with the powerful learning capacity of the FNN, DP-RFECV-FNN demonstrates the ability to identify both known and novel malware types and achieves higher accuracy while maintaining strict privacy controls compared with recent papers. DP-RFECV-FNN achieves an accuracy ranging from 97.78% to 99.21% while utilizing static features and 93.49% to 94.36% for dynamic features of Android applications to detect whether it is malware or benign. These results are achieved under varying privacy budgets, ranging from $\epsilon = 0.1$ to $\epsilon = 1.0$. Furthermore, our proposed feature selection pipeline enables us to outperform the state-of-the-art by significantly reducing the number of selected features and training time while improving accuracy. To the best of our knowledge, this is the first work to categorize Android malware based on both static and dynamic features through a privacy-preserving neural network model.

1. Introduction

As the Internet of Things (IoT) continues to grow rapidly, the security of IoT networks becomes a concern. IoT devices, e.g., smart watches, that are controlled or powered by Android serve as key components within interconnected IoT environments [1]. The use of the Internet of Medical Things (IoMT) in e-healthcare has significantly grown, enhancing patient monitoring and efficiency in medical care through wireless and wearable technologies [2]. IoMT comprises a variety of healthcare devices connected to the Internet, such as medical scanners, smart thermometers, and biosensors that can be either worn or implanted in the body. The Android operating system is widely used in these devices as the main operating system, making it crucial to verify that the Android apps in use are safe and not malicious [3]. According to [4], around 70% of IoT devices employ the Android operating system for data exchange and communication. Fig. 1 illustrates various application areas that utilize Android applications within IoT networks [5]. Consequently, malware attackers frequently attempt to hack those applications to steal patient data, which could threaten the security of the entire healthcare network. Ensuring the security of these Android applications is crucial for keeping the network of healthcare devices safe. Therefore, the detection of Android malware is of utmost importance, as it directly impacts the protection of sensitive patient

* Corresponding author.
E-mail addresses: fnawshin@qu.edu.qa (F. Nawshin), dunal@qu.edu.qa (D. Unal), m.hammoudeh@kfupm.edu.sa (M. Hammoudeh), p.n.suganthan@qu.edu.qa (P.N. Suganthan).

https://doi.org/10.1016/j.adhoc.2024.103523
Received 13 February 2024; Received in revised form 4 April 2024; Accepted 21 April 2024
Available online 25 April 2024

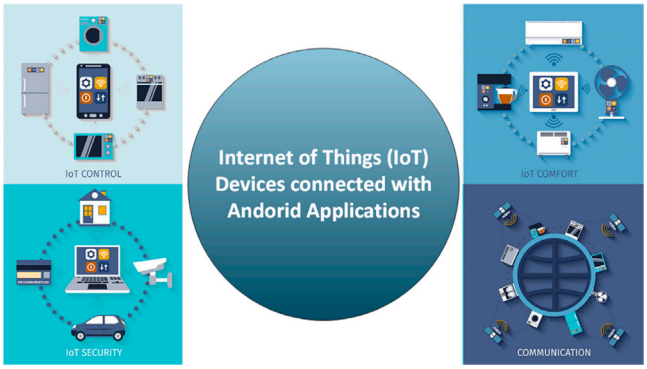


Fig. 1. Interconnection of Android devices and applications with IoT networks [5].

information and the overall integrity of the IoT networks in the medical sector [6]. The healthcare applications include data containing medical records, treatment plans, and health monitoring details. If this information were exposed or compromised due to inadequate privacy measures in malware detection processes, it could lead to significant privacy violations. By implementing DP, the malware detection method can effectively detect threats within Android applications while ensuring that sensitive health information remains secure and private during the training processes.

Android devices, which constitute a large percentage of the IoT network, are susceptible to security threats and malware attacks. Several machine learning (ML) and deep learning (DL) based algorithms were developed by researchers for real-time analysis of the security of Android IoT devices to identify malicious activities [1]. AI-powered malware analysis can help IoT networks improve zero trust security. Malware analysis is integral to enhancing zero trust security in IoT networks as it provides insights that improve various aspects of their security posture. Amin et al. [7] presented a deep learning system designed to detect malware in IoMT and smartphone applications, focusing on Android OS. It introduced a novel feature detector based on deep learning that can be easily trained and utilized with various classifiers to assess an application's behavior as benign or malicious. Their proposed system achieved high accuracy and highlighted its potential to significantly improve the security of IoT and smartphone ecosystems against malware threats. However, this approach did not consider the privacy of user-sensitive data, such as healthcare data, during the training phases. There was no privacy-preserving approach used to secure training data. Our DP-RFECV-FNN framework addresses this gap by incorporating DP with an FNN to safeguard against data poisoning attacks. By introducing noise into the training dataset, our model prevents attackers from exploiting the data while maintaining high performance metrics. Additionally, the feature selection method used in our approach is particularly effective for managing large datasets. Moreover, our model uniquely categorizes malware types which was also not explored in the previous work.

The intersection between Android malware detection and AI-enabled zero trust is related to malicious applications on an Android device activating potential security measures that ultimately prevent the spread of malware within the IoT network. The majority of the state-of-the-art malware detection methods utilize centrally located data samples for training, which can lead to the leakage of personal or sensitive information. The attackers can obtain sensitive information by analyzing the applications directly, which leads to the violation of users' privacy. Differential Privacy (DP) [8], Homomorphic Encryption (HE) [9], and Secure Multi-Party Computation (SMPC) [10] are some of the approaches developed to integrate with ML and DL based approaches to protect sensitive information and users' privacy while enabling effective model training and data analysis. This article utilizes DP as a privacy-preserving approach in training the FNN model to detect and classify Android malware using both static and dynamic features to enhance zero trust security in IoT networks.

Our proposed approach is distinctive by its ability to be implemented on-device, eliminating the need for centralized ML models hosted on external servers or cloud platforms. This decentralized approach aligns seamlessly with the principles of zero trust architecture. In traditional cybersecurity models, trust is often placed in external entities, such as cloud providers, to conduct ML on sensitive data. In contrast, our approach distributes the ML process across end-user devices, ensuring that sensitive information remains localized. To ensure that our approach follows zero trust security principles, each Android application of the system is authenticated and verified before getting access to the IoT network. Prior to this verification, it is also mandatory to check the status of the application, i.e., whether it is benign or malicious. Only the benign applications will proceed to the authentication and verification step. The malicious applications are not granted access to the system and will be investigated further to classify the malware category. As every APK needs to be examined every time, considering resource-constraints of IoT devices in the long run, the architecture of our FNN model is optimized for IoT environments, ensuring that it is lightweight enough to perform efficiently on power-constrained devices. Furthermore, the feature selection method we used is capable of identifying the most relevant features from a large feature vector that reduces the number of features significantly without degrading the model's performance. This leads to speeding up the training time, which ensures our model remains lightweight and efficient with limited power resources. DP-RFECV-FNN employs both static and dynamic analysis to detect malicious applications that try to look as benign. Static analysis examines the code, permissions, and manifest files without running the applications, while dynamic analysis analyzes the runtime behavior of an application, such as network traffic and system calls, that helps to find unusual behaviors. Because of utilizing dynamic analysis, our solution is capable of identifying malware that tries to hide by giving false information. Consequently, it ensures only benign applications will be permitted to enter the system. Fig. 2 illustrates our proposed framework featuring the zero trust security model.

The contributions of this article are summarized below.

1. We propose a privacy-preserving solution, called DP-RFECV-FNN, which incorporates DP and a FNN model to detect and classify Android malware using both static and dynamic features. DP-RFECV-FNN is designed to meet and enforce zero trust in IoT networks.
2. DP-RFECV-FNN is capable of working with large datasets characterized by a large number of features. We combined three feature selection methods: Feature Importance, Pearson Correlation Coefficient (PCC), and Recursive Feature Elimination with Cross-Validation (RFECV) techniques to select the most relevant features, which helps reduce the time complexity of the model significantly. We evaluated this feature selection procedure on two different datasets, including static and dynamic features, and obtained promising performances in both cases compared with the methods employed in previous research papers.
3. Our work is the first to integrate both static and dynamic features in Android malware detection and classification using DP, while aligning with the principles of zero trust security. Static features offer valuable insights into the structural aspects, permissions, and other inherent characteristics of applications, while dynamic analysis ensures a more robust and accurate detection approach through real-time observations of application behavior. DP-RFECV-FNN outperformed the privacy-preserving method (within the same privacy budget) and surpassed the results of the non-privacy-preserving approach compared to [11]. The experiments were conducted on the CCCS-CIC-AndMal-2020 dataset for static features and the CICMalDroid 2020 dataset for dynamic features, achieving an accuracy of 97.84% and 94.43%,

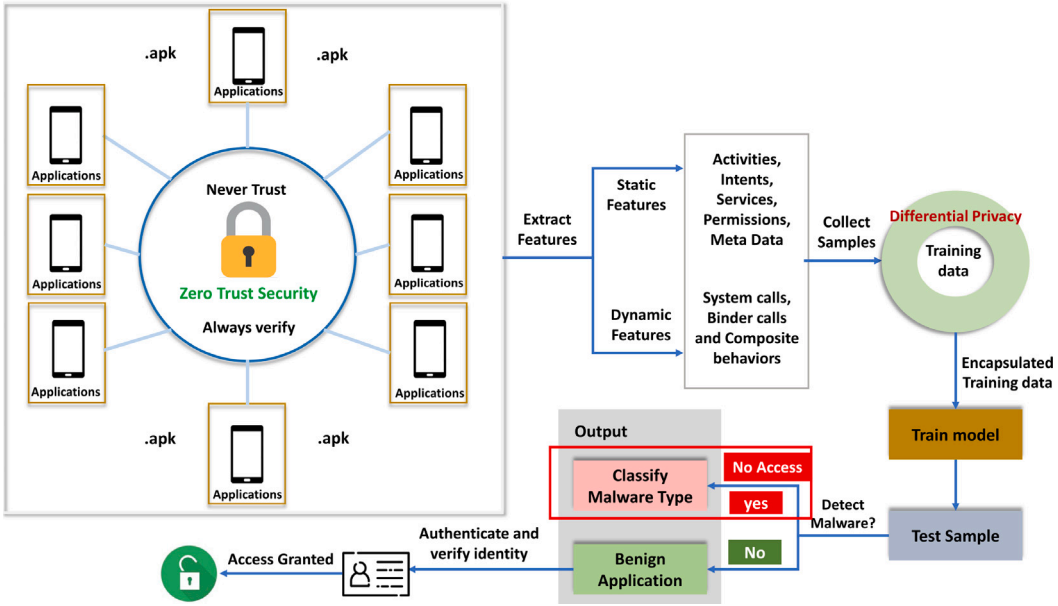


Fig. 2. Our proposed framework featuring zero trust security.

respectively, under a privacy budget of $\epsilon = 0.5$ in detecting malicious Android applications. Furthermore, our method achieved an accuracy of 92.09% using static features and 89.54% using dynamic features within the same privacy budget for classifying malicious Android applications.

The remainder of the article is organized as follows. We briefly explain the zero trust security architecture and DP Preliminaries in Section 2. We present the related work in detecting IoT malware and Android malware in Section 3. We describe the methodology of DP-RFECV-FNN, in Section 4. We present the evaluation of the DP-RFECV-FNN in Section 5. Discussion and future work are discussed in Section 6. Finally, we conclude the paper in Section 7.

2. Zero trust security architecture and differential privacy

This section gives the necessary preliminaries of zero trust and DP with the mathematical formulations. It aims to help the readers understand how zero trust security and DP are integrated into DP-RFECV-FNN.

2.1. Zero trust and Android malware analysis

Malware analysis on Android devices is critical for understanding and countering malicious software targeting the Android operating system. On the other hand, zero trust is a security model that challenges the traditional notion of trusting entities inside a network and instead emphasizes continuous verification and validation of all devices, users, and applications trying to access resources, regardless of their location [12]. The link between malware analysis on Android devices and zero trust security is rooted in their shared objective of strengthening cybersecurity by continuously verifying, monitoring, and adapting security measures based on the ever-expanding threat vector and potential risks. Additionally, malware analysis contributes to enhancing endpoint security on Android devices. In a zero trust environment, endpoints (IoT devices) are assumed untrusted, and robust endpoint security is essential. Malware analysis insights help design and implement security measures at the endpoint level. However, malware analysis usually involves handling sensitive information, and zero trust emphasizes data protection and privacy. Aligning both approaches ensures that the analysis is conducted in a privacy-preserving manner, critical in the context of evolving privacy regulations. Finally, zero trust underscores the need for dynamic access controls based on real-time security assessments. The insights from malware analysis into the evolving threat landscape on Android devices, allow for the dynamic optimization of access controls to prevent or mitigate potential risks.

The widespread adoption of Android operating system in IoT devices has rendered them attractive targets for malicious actors. In response to this rising threat landscape, researchers have continuously proposed a multitude of solutions and algorithms employing ML and DL to effectively identify these malicious applications. The mission for robust detection mechanisms encompasses the integration of both static and dynamic features in the analysis of Android applications. Static features encompass a range of factors such as permissions sought by applications, API calls, metrics detailing code structure and complexity, meta data embedded in the Manifest file, as well as strings and keywords present in the application's code [13]. On the dynamic front, features extend to network communications, runtime application behavior, system calls, permissions requested during runtime, and the interaction of applications with users [14]. This intensive effort towards leveraging both static and dynamic features emphasizes the comprehensive nature of contemporary approaches in identifying and mitigating malicious applications on Android devices.

2.2. Differential privacy

Sensitive information can be protected via the application of Differential Privacy (DP) [15], an approach that facilitates meaningful data analysis while ensuring the confidentiality and privacy of sensitive data. DP introduces a controlled level of noise to the data, achieving a balance between extracting valuable insights and preventing the disclosure of specific details about individual data points. This approach not only improves the security of sensitive information but also enables organizations to gain actionable intelligence from their datasets without compromising the privacy of the individuals contributing to the dataset.

DP protects user privacy by obscuring the visibility of any single data entry, ensuring individual data points are indistinguishable, yet still allows for the collection of meaningful information from the overall dataset. The inclusion or exclusion of any single data does not affect the result of data analysis. In DP, a controlled and random noise is added to the training data that prohibits attackers from inferring sensitive information about individuals from the training dataset.

2.3. Privacy budget

Privacy Budget is often used to express the level of privacy protection and the amount of required noise that is to be added to the data. ϵ and δ are the two parameters to denote the privacy budget in DP. The lower the value of ϵ, the more noise needs to be added to the data, which indicates a stronger privacy guarantee. It quantifies the impact of a single individual's data on the privacy of the overall computation. δ is another privacy parameter that denotes the amount of deviation that a differentially private mechanism allows for. A smaller value of δ indicates a higher probability of maintaining privacy guarantee by the mechanism [16]. The output ψ of an arbitrary algorithm M for any two adjacent datasets D and D' satisfies

$$\Pr[M(D) \in \psi] \leq e^{\epsilon} \Pr[M(D') \in \psi] + \delta \tag{1}$$

Eq. (1) ensures privacy by expressing that the probability of an algorithm's output on dataset D being in a certain set ψ is not much higher than the probability on a slightly different dataset D'. It also demonstrates a trade-off between privacy (ϵ) and flexibility (δ). A smaller value of ϵ indicates stricter privacy, whereas a larger value of δ means more flexibility in the system. It is crucial to make a balance between these two privacy parameters to implement DP as well as allow useful data analysis.

Given a function $f : D \to \mathbb{R}^d$, for any two adjacent datasets D and D', define the global sensitivity of f as

$$\Delta f = \max_{D,D'} \|f(D) - f(D')\|_2 \tag{2}$$

where $\mathbb{R}^d$ represents a d-dimensional real vector, and $\|f(D)-f(D')\|_2$ is the L2 distance between $f(D)$ and $f(D')$. D and D' are nearly identical except for a minimal change in at most one data point [17].

For any two adjacent datasets, D and D', Eq. (2) calculates the maximum L2 distance between the outputs of the functions. It measures the maximum change in the output of the function when transitioning between neighboring datasets which provides a computation on the sensitivity of the function to changes in the input data.

2.4. Gaussian mechanism

The Gaussian mechanism, denoted as $M(D)$, is a privacy-preserving mechanism applied to a function f operating on dataset D. It introduces noise sampled from a Gaussian distribution to the output of f to protect the privacy of individual data points. The Gaussian mechanism with parameter δ is defined as

$$M(D) = f(D) + \text{Noise} \tag{3}$$

where Noise $\sim \mathcal{N}(0, \sigma^2 I)$ satisfies the Gaussian distribution.

The noise term, represented as Noise, follows a Gaussian distribution with mean 0 and covariance matrix $\sigma^2 I$, where I is the identity matrix. The expression for σ^2 is given by

$$\sigma^2 = \frac{2\Delta^2 \log\left(\frac{1.25}{\delta}\right)}{\epsilon^2} \tag{4}$$

where Δ is the sensitivity of the function. The parameter σ^2 determines the variance of the added noise and is crucial for achieving DP. The value is computed based on the sensitivity (Δ) of the function and the desired privacy parameters ϵ and δ [18].

3. Related work

While developing our research on employing the DP-RFECV-FNN model for Android malware detection, we followed a comprehensive review of relevant literature. This review focused on selecting papers that were directly related to our research and emphasized papers that used static and dynamic analyses for malware detection. Our selection included papers addressing malware threats in both IoT and Android environments, with an emphasis on those that utilized deep learning approaches. Additionally, we prioritized papers that incorporated DP within this domain so that we could compare them with our efforts.

Bendiab et al. [19] presented a novel approach to IoT malware traffic classification using visual representation and DL. It focused on detecting malicious network traffic at the package level, comprising a dataset of 1000 .pcap files of normal and malware traffic. The methodology employed the Residual Neural Network (ResNet50) for analyzing visual representations of network traffic. The results showed a 94.50% accuracy rate in malware traffic detection. However, there is a need to include extensive and diverse datasets to enhance the predictive performance of the model. The experiment did not extensively cover the performance of the system in a real-time environment, which is crucial for practical deployment.

A DL-based method for detecting IoT malware through system calls was presented in [20]. Utilizing the Recurrent Neural Network (RNN) classified system calls into benign and malicious categories. The dataset comprised IoT malware samples from IOTPOT and benign samples from Ubuntu system files. Their method achieved an accuracy of 97.72% in malware detection. However, they focused only on the detection of IoT malware using system calls and did not include malware category classification, which could have provided a more detailed analysis of different malware types like viruses, trojans, worms, and botnets.

Ali et al. [21] proposed a multitask classification using an LSTM-based DL model for IoT malware detection and identification. Their approach aimed to enhance IoT security by classifying and identifying various types of malware attacks. Their methodology used data from 18 different IoT devices divided into flow, flags, and packets for feature selection. A time-series analysis of the traffic flows was performed during the experiment, and the model achieved an accuracy between 88.45% and 95.83%. However, they used only a laboratory-based dataset to train and evaluate the performance of the model. They did not include implementing and testing the model in real-world IoT networks to evaluate its practical applicability.

In [22], Chaganti et al. introduced a DL-based Bidirectional-Gated Recurrent Unit Convolutional Neural Network (Bi-GRU-CNN) model for IoT malware detection and classification. This approach used Executable and Linkable Format (ELF) binary file byte sequences as input features. However, their research only used ELF binary files, which limited its applicability to other file formats prevalent in IoT devices. They did not explore the adaptability of the model to other file formats for comprehensive IoT security.

Many research efforts applied DL for malware detection on Android devices. Lu et al. [23] used the combination of a Deep Belief Network (DBN) and Gated Recurrent Unit (GRU) to develop a method for Android malware detection. They used both static and dynamic features where DBN processed static features and GRU processed dynamic features extracted from Android applications. They compared their model against state-of-the-art ML models and achieved an accuracy of approximately 96%. However, the number of malicious samples they used in the experiment was not enough. Besides, the time consumption of their method was larger than that of traditional ML methods.

Zhang et al. [24] presented TC-Droid, a framework for Android malware detection using text classification methods. The approach used the convolutional neural network (CNN) to extract and select features automatically, bypassing the need for manual feature engineering. TC-Droid demonstrated better performance over existing models (NB, LR, KNN, RF) in terms of accuracy and precision. However, they focused only on static analysis, excluding dynamic analysis, which could provide additional insights into malware behavior. The exclusion of dynamic analysis limits the comprehensiveness of malware detection. Furthermore, they did not include how the proposed model might adapt to new and emerging threats.

MAPAS [25] used CNN and API call graphs to analyze the behavior of malicious Android applications. CNN was used only to identify common features from Android applications, such as API call graphs.

Similarly, Elayan et al. [26] proposed another DL-based Android malware detection using a Gated Recurrent Unit. They used both API calls and permissions from the applications as static features during the experiment. They evaluated the model against the CICAndMal2017 dataset and achieved an accuracy of 98.2%. However, these papers relied only on static analysis, which is not able to capture the complex behaviors of advanced malware.

DL and the Rock Hyrax Swarm Optimization method, namely RHSODL-AMD, were combined in [27] to detect Android malware. They extracted API calls and the most significant permissions from Android applications to differentiate between malware and benign applications. Besides, they used Attention Recurrent Autoencoder (ARAE) and Adamax optimizer to detect Android malware. They evaluated the proposed model only on the Andro-AutoPsy dataset and achieved an accuracy of 99.05%. This is one of the shortcomings of their method in that it used only one dataset, which limits the generalization of the model to other types of malicious applications. Additionally, while the model showed high accuracy, there was a lack of discussion on its performance in terms of false positives and false negatives, which are critical in malware detection.

Malware detection using privacy-preserving ML gained a lot of attention in recent years. Gálvez et al. [28] developed a privacy-preserving framework called LiM ('Less is More') for classifying Android malware using Federated Learning with 200 users and 50 rounds of federation where the information about the installed applications do not leave the local devices. Their solution achieved a 95% F1 score and used the MaMaDroid dataset for the experiment. However, they did not explore the scalability of LiM in a real-world scenario with an extensive number of clients and malware samples. While the paper addressed privacy concerns, it did not extensively discuss the potential trade-offs between privacy and detection accuracy.

Jiang et al. [29] proposed a framework for an Android malware classifier using federated learning named FedHGCDroid. Their model incorporated convolutional and graph neural networks for accurate malware detection. They used the Androzoo dataset for the experiment and achieved an accuracy of 91.3% in Android malware detection and 83.39% in malware classification. However, it primarily focused on static features only and did not include dynamic features in malware detection and classification.

DNNdroid [30] is another federated learning solution for malware detection to address the issues of free malicious applications of Android and their reliance on permissions. In [30], input from all users was collected simultaneously to enhance the model without revealing specific user data. Their model achieved an F1 score of 97.8% with a client recall rate exceeding 95% and a false positive rate below 0.95. Deng et al. [11] proposed a malware detection framework, namely MDHE, using another privacy preservation approach, DP, for IoT networks. They used CapsNet as the learning model and only used static features such as permission and API features for the experiment. In their approach, features were perturbed, meaning that noise was added to the features of the training data before it was trained. This is one of the drawbacks of this approach because an adversary can learn sensitive information by analyzing the perturbed features of individual data points, which affects the privacy of the training data. The adversary refers to a malicious entity or attacker whose target is to compromise privacy by extracting sensitive information from the training dataset. The authors only evaluated their model in detecting malware and did not consider classifying malware types. In addition, their proposed solution did not include dynamic features, which is another shortcoming of their model because dynamic analysis is adaptable to new threats and allows for real-time detection of malicious activities.

Our proposed framework addressed the drawbacks of these approaches and proposed a comprehensive solution of integrating DP and FNN using both static and dynamic features of Android applications aligned with the principles of the zero trust model. We evaluated our model on multiple datasets and included an extensive analysis of trade-offs between privacy and utility under different privacy budgets in both malware detection and classification. Within the same privacy budget, the DP-RFECV-FNN framework demonstrated superior performance in malware detection compared to the approach outlined in [11].

Table 1 compares existing methods with our proposed framework in Android malware detection and classification.

4. Methodology and implementation

DP-RFECV-FNN is initiated by loading a malware dataset containing static/dynamic features and labels. DP-RFECV-FNN comprises two parts. The first part includes the detailed feature selection procedure. The second part contains the design of the proposed model, including designing the neural network model, and defining and incorporating DP mechanisms in the training process. Fig. 3 demonstrates the flow diagram of the proposed model.

4.1. Security assumptions and threat model

This section outlines the foundational security assumptions that support our approach and details the threat model we have developed to guide the implementation.

4.1.1. Security assumptions

Our approach to Android malware detection through the DP-RFECV-FNN model within IoT networks is composed of several assumptions that are essential for comprehending its operational effectiveness and scope:

- Considering the inherent vulnerabilities within the IoT network, the model employs a zero trust framework, which operates on the principle of "never trust, always verify". This approach is appropriate given the heterogeneity and complexity of IoT networks, which include a broad range of devices with varying capacities.
- The security perimeter is considered dynamic, evolving in response to real-time threat assessments. Security policies and measures are reconfigured based on continuous monitoring of the behavior of the devices and network interactions to incorporate the zero trust principle and least privilege access.
- The model assumes the existence of sophisticated adversaries capable of deploying advanced malware and evasion techniques to avoid traditional security mechanisms.
- The model focuses on keeping user information safe and private. It uses special privacy methods, DP, to make the data anonymous by adding noise to the training data. DP makes it difficult for attackers to infer training examples or introduce targeted poisoning attacks.

4.1.2. Threat model

To strengthen these assumptions, the threat model for the DP-RFECV-FNN describes the primary security risks and adversarial actions predicted in IoT environments, in alignment with zero trust principles [37]:

- The foremost threat anticipated is the spread of malware across the IoT network with a variety of malware categories such as spyware, ransomware, and worms. Using zero trust principles, we plan to block unauthorized access and stop malware from spreading by imposing strict access control and constantly monitoring the network.
- A serious threat involves outsiders getting unauthorized access to sensitive data and possibly stealing out. By using the security strategies of zero trust [37], our approach works to protect data, whether it is stored or being sent, using encryption and strict access rules that are based on real-time risk assessments.

Table 1
Comparison between malware detection and classification methods.

Method	Category	Feature	Malware detection	Malware classification	Learning model	Privacy protection
[31]	Dynamic analysis	System calls	✓	✗	DEEPMALWAR	✗
[32]	Dynamic analysis	Malware image features	✓	✗	DPNSA	✗
[33]	Dynamic analysis	System calls, binders and composite behavior	✓	✓	Stacked ensemble machine learning	✗
[34]	Dynamic analysis	API calls	✓	✗	Bi-LSTM	✗
[35]	Static analysis	Intents, Permissions, API calls	✓	✗	CNN	✗
[36]	Static analysis	Permissions, API calls	✓	✗	SOMDROID	✗
[11]	Static analysis	Malicious subgraph	✓	✗	CapsNet	✓
DP-RFECV-FNN	Static analysis Dynamic analysis	Activities, Intents, Services, Permissions, Meta data System calls, Binder calls and Composite behaviors	✓	✓	DP-RFECV-FNN	✓

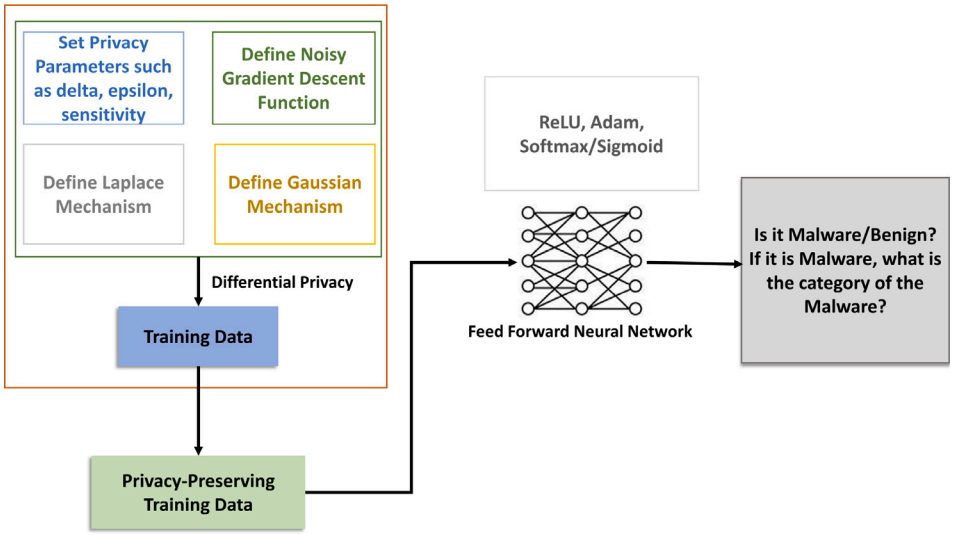


Fig. 3. Flow diagram of DP-RFECV-FNN.

- Given the resource-constrained characteristics of many IoT devices, they are particularly susceptible to DoS attacks, whose goal is to overwhelm the processing capabilities of the devices. The model incorporates adaptive security policies that dynamically allocate resources to essential services and functions for maintaining operational integrity.
- Considering the risk of insider threats or the compromise of previously trusted devices, the zero trust framework within the model advocates for continuous verification of all devices and network traffic. This requires regular re-assessment of trust levels and access rights to ensure that only authorized entities can access system resources.

4.2. Dataset details

We used the CCCS-CIC-AndMal-2020 dataset [38,39] for analyzing the static features and the CICMalDroid 2020 dataset [40,41] for dynamic features. AndroidManifest.xml file was analyzed, and the static features such as Activities, Broadcast receivers and providers, Permissions, and System features were extracted from this file. CopperDroid [42], a VMI-based dynamic analysis system, was used to analyze the dynamic features, and system calls, binder calls, and composite behaviors were extracted from .apk files as dynamic features. Table 2 displays the dataset details for static and dynamic analysis. We compiled the static analysis dataset by gathering nearly 15,000 benign APK files and 15,000 malicious APK files that contain various malware categories, including Adware, SMS, Ransomware, Riskware, and Trojan. After that, we collected around 1800 benign APK files and 9800 malicious APK files for dynamic analysis, containing malware categories such as Adware, Banker, SMS, and Riskware.

4.3. Feature selection

Learning models with too many features are more likely to overfit the training data because they capture the noise rather than the true patterns of the training data, and eventually, this degrades the performance of the learning model. During the process of feature selection, we selected some baseline feature selection methods initially used in malware detection, such as Mutual Information [43], Variance Threshold [44], and Pearson Correlation Coefficient (PCC) [45], and evaluated the performance of the model. The number of features was also noted along with the performance metrics. Tables 3 and 4 show the comparison of feature selection methods employed in DP-RFECV-FNN with other existing methods using CCCS-CIC-AndMal-2020 and CICMalDroid 2020 dataset respectively. The resulting data highlighted that the feature selection technique used in DP-RFECV-FNN not only yielded a higher accuracy but also achieved a high level of precision, recall, and F1 Score across both datasets. It achieved higher performance metrics with a significantly smaller feature set, with only 19 features for the CCCS-CIC-AndMal-2020 dataset and 33 for the CICMalDroid 2020 dataset. DP-RFECV-FNN combines three feature selection techniques (Feature Importance Score, PCC and RFECV) described below to come up with the most relevant features out of a large dataset. The feature selection procedure is shown in Fig. 4. The top 15 selected features are given in Table 5.

Feature importance score. The feature importance score was calculated to quantify the contribution of each feature to the dataset. This score plays a vital role in identifying the set of features that have a stronger impact on the output of the model. In addition, the feature importance score helps to select the relevant features and interpret and understand the underlying patterns of the data. Feature importance score was

Table 2
Dataset details for static and dynamic analysis.

Analysis type	Application type	Dataset	Features	Samples	Malware type	Samples
Static	Malware	CCCS-CIC-AndMal-2020 [38,39]	Activities, Intents, Services, Permissions, Meta data	14,988	Adware SMS Ransomware Riskware Trojan	2988 3000 3000 3000 3000
	Benign			14,943	**Total**	**14,988**
	Total number of samples			**29,931**		
Dynamic	Malware	CICMalDroid 2020 [40,41]	System calls, Binder calls, Composite behaviors	9803	Adware Banker SMS Riskware	1253 2100 3904 2546
	Benign			1795	**Total**	**9803**
	Total number of samples			**11,598**		

Table 3
Comparison with other feature selection method using CCCS-CIC-AndMal-2020 dataset.

	Accuracy	Precision	Recall	F1-Score	Number of features
Mutual information	0.991	1	1	0.99	1283
Variance threshold	0.976	0.98	0.98	0.98	2326
PCC	0.987	0.99	0.99	0.99	2248
DP-RFECV-FNN	**0.997**	**1**	**1**	**1**	**19**

Table 4
Comparison with other feature selection method using CICMalDroid 2020 dataset.

	Accuracy	Precision	Recall	F1-Score	Number of features
Mutual information	0.935	0.86	0.91	0.88	336
Variance threshold	0.947	0.91	0.88	0.90	470
PCC	0.95	0.95	0.95	0.95	428
DP-RFECV-FNN	**0.958**	**0.96**	**0.96**	**0.96**	**33**

calculated for all the features and we removed the features that have a feature importance score of 0 and retained the remaining features in the first step of the feature selection method. The Random Forest classifier was used in the experiment to measure the feature importance score by calculating how much every feature within each tree in the forest helps make a better decision and sums all this up to the total score [46].

Pearson Correlation Coefficient (PCC). During the second step of the feature selection method, the PCC between each feature and the target variable (benign or malicious) was calculated on the feature set obtained from the first step. This helps to determine the features that have a strong correlation with the target variables. After applying PCC, some features that had a weak correlation with the target variable ($threshold = 0.2$) are eliminated [47], and the remaining features were stored for the further step.

Recursive Feature Elimination with Cross-Validation (RFECV). To identify the most relevant features from the remaining list obtained from the second step, the RFECV technique was applied as a final step of the feature selection process. RFECV starts performing on the selected dataset by iteratively eliminating features one at a time, and then the performance of the model is evaluated through cross-validation at each step [48]. Random Forest was used as a classifier to select the best features for a model. It uses many decision trees to rank the importance of different features [49]. Finally, this optimal subset of features is obtained and observed that the training time was potentially improved. To address the class imbalance of the dynamic dataset, the Synthetic Minority Over-sampling Technique (SMOTE) is applied specifically to the training set, augmenting the minority class instances. Furthermore, the features are standardized using the StandardScaler to ensure uniform scaling and improve convergence during training.

4.4. Data preprocessing

Data preprocessing is applied to the raw dataset to address noisy data such as missing values, outliers, and errors, which affects the performance and accuracy of learning models. To ensure that the ML model is trained on reliable and accurate data, The following steps are followed to prepare and preprocess the dataset before feeding it into the model.

Data cleaning. Tomek links [50] is used to identify the noisy and borderline samples. It helps to address the samples that are close to each other but belong to different classes. These ambiguous samples are identified with the help of Tomek links and removed to eliminate misclassification. This process results in a cleaner dataset, reducing noise and improving class separation.

Data shuffling. Before splitting into training and testing data, the whole dataset is shuffled to ensure that both sets have a representative mix of samples. Data shuffling is a crucial step in working with ML models. If the data is ordered, then the models can learn the patterns based on the ordered dataset instead of learning the inherent properties of the data, which affects the performance and accuracy of the model. Data shuffling helps to get rid of over-fitting and ensures that the model generalizes well to unseen data.

Feature normalization and scaling. The Min–Max scaling method is used to scale all the features between 0 and 1. Large differences in feature values can lead to numerical instability, which causes issues like overflow or underflow. Normalization helps to prevent these issues and ensures numerical stability during computation. Regularization techniques such as L1 and L2 regularization penalize large coefficients. Scaling features help to maintain a balance on the impact of regularization across all the features and prevent the model from unfairly penalizing variables with larger values [51]. Fig. 4 shows the flow diagram and Algorithm 1 demonstrates the feature selection procedure and data preprocessing method employed during the experiment.

Dataset splitting. Various combinations of training, testing, and validation sets are carried out to determine the optimal combination. The performance of the proposed model is observed in various combinations of training, testing, and validation sets, and the performance was almost the same for each variation. The dataset is partitioned into 70% for training, 10% for validation, and 20% for testing purposes. While training, the training set is used, and the validation set is used for tuning the hyperparameters. Finally, the training and validation sets are combined for training, and a test set is used to measure the performance of the learning models.

Table 5
Top 15 static and dynamic features.

Static features	Dynamic features
Dataset Name: CCCS-CIC-AndMal-2020 [38,39]	Dataset Name: CICMalDroid2020 [40,41]
com.fb.iwidget.ActionReceiver	getReceiverInfo
com.fb.iwidget.OverlayActivity	getActivityInfo
com.batch.android.BatchActionService	mprotect
android.permission.CALL_PHONE	ftruncate64
com.fb.iwidget.MainService	CREATE_FOLDER___
android.permission.ACCESS_NETWORK_STATE	brk
com.fb.iwidget.action.SHOULD_REVIVE	sigprocmask
android.intent.category.DEFAULT	unlink
com.fb.iwidget.ExpandWidgetProvider	FS_ACCESS(CREATE_READ_WRITE)
com.fb.iwidget.PreferencesActivity	FS_ACCESS(CREATE_WRITE)_
com.fb.iwidget.MainActivity	fdatasync
com.fb.iwidget.SnapAccessService	stat64
android.intent.category.BROWSABLE	rename
android.intent.action.VIEW	pwrite64
android.permission.BIND_ACCESSIBILITY_SERVICE	getApplicationRestrictions

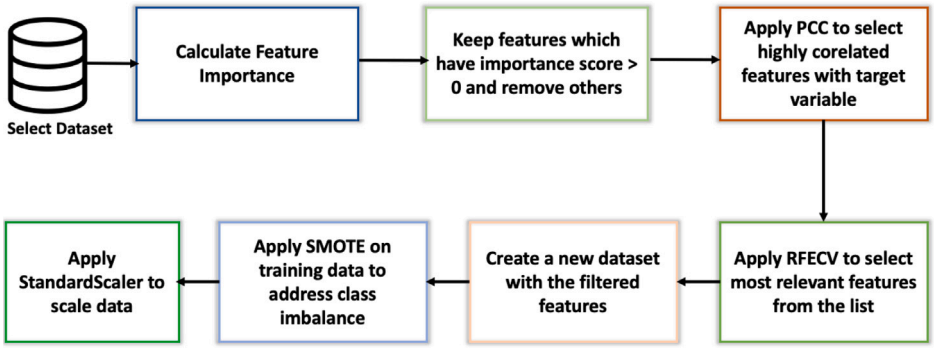


Fig. 4. Flow diagram for feature selection and data preprocessing.

Algorithm 1 DP-RFECV-FNN Feature Selection and Data Preprocessing.

Load the malware dataset
Feature selection and data preprocessing:
Calculate the feature importance score for all the features:
Feature_Imp_Score = f(All Features)
Remove features with an importance score of 0:
features_init = {feature | Feature_Imp_Score(feature) > 0}
Apply PCC to select features with a strong linear correlation (ρ) with the target variable:
features_pred = {feature | ρ(feature, target) > threshold}
where ρ is calculated on features_init
Apply RFECV using Random Forest on the features list obtained from step 5:
features_final = RFECV(features_pred)
Create a dataset with the selected features:
TempDataset = Original Dataset[:, features_final]
Use SMOTE to address class imbalance in training data:
Balanced Dataset = SMOTE(TempDataset)
Standardize or scale the data using StandardScaler:
Final Dataset = StandardScaler(Balanced Dataset)

4.5. Integrating privacy and zero trust into the model

Privacy parameters. In this step, critical privacy parameters are defined. Sensitivity (Δ) denotes the maximum amount by which the output of the model could change with the alteration of a single data point. Delta (δ) represents the privacy parameter related to the probability of a privacy breach and is calculated based on the size of the dataset. The number of classes in the dataset (C) is determined, and the total number of training iterations is set to 30.

Epsilon (ϵ) values are explicitly defined to create a range of privacy parameters for testing that controls the level of privacy protection. By experimenting with different ϵ values, we explore the trade-off between model accuracy and privacy preservation by the algorithm.

DP mechanisms. DP mechanism is introduced in this part. The amount of noise is determined by the sensitivity of the gradients, privacy parameters (ϵ) and δ, for additional privacy control. The Gaussian mechanism is used to add Gaussian noise to gradients, considering sensitivity and privacy parameters. The Laplace Mechanism function is used to insert Laplace noise into the count of the data. In this step, the Gradient Function utilizes TensorFlow's gradient tape mechanism to record operations for automatic differentiation. Gradients of the parameters of the model are calculated concerning a defined loss function. Then, another function, called the L2Clip function, is used to ensure that the gradients do not exceed a specified L2 norm and contribute to the privacy-preserving nature of the training process. Algorithm 2 states the functions for both Laplace and Gaussian mechanisms.

Neural network model. In this step, a neural network model is constructed using TensorFlow/Keras. The model architecture comprised an input layer, two hidden layers with 64 and 32 neurons and Rectified Linear Unit (ReLU) activation functions, and an output layer with sigmoid and softmax activation for prediction and multi-class classification, respectively. The model is compiled with the Adam optimizer and sparse categorical cross-entropy loss function. In developing the model, an iterative process is used for experimenting with various combinations of input and output layers to optimize performance. The DP-RFECV-FNN configuration was finalized after comprehensive testing, and it showed an optimal balance between privacy preservation and detection performance. Table 6 shows the performance of different combinations of FNN architecture on malware detection while $\epsilon = 0.5$ using CCCS-CIC-AndMal-2020 dataset.

Table 6
Performance comparison of FNN architectures with varying layers and neurons at $\epsilon = 0.5$.

FNN configuration	No. of layers	Layer 1 neurons	Layer 2 neurons	Accuracy	Precision	Recall	F1 Score
1	2	32	32	0.954	0.96	0.95	0.95
2	2	64	64	0.973	0.97	0.97	0.97
3	2	64	128	0.965	0.97	0.97	0.97
4	2	32	128	0.973	0.97	0.97	0.97
5	2	128	128	0.974	0.98	0.97	0.97
6 (DP-RFECV-FNN)	**2**	**64**	**32**	**0.978**	**0.98**	**0.98**	**0.98**

Algorithm 2 DP mechanisms: Laplace and Gaussian Noise

function LaplaceMechanism(sample_count, Δ, ϵ)
  scale $\leftarrow \frac{\Delta}{\epsilon}$
  noise $\leftarrow$ LaplaceNoise(0, scale)
  return sample_count + noise
end function
function GaussianMechanism(gradients, Δ, ϵ, δ)
  noisy_gradients $\leftarrow$ []
  num_samples $\leftarrow$ length of gradients
  scale $\leftarrow \frac{\Delta}{\epsilon}\sqrt{2\ln\left(\frac{1.25}{\delta}\right)}$
  for gradient in gradients **do**
    shape $\leftarrow$ shape of gradient
    noise $\leftarrow$ GaussianNoise(0, scale, size = shape)
    noisy_gradients.append(gradient + noise)
  end for
  return noisy_gradients
end function
function Gradient(model, inputs, targets)
  loss $\leftarrow$ SparseCategoricalCrossentropy(targets, model(inputs))
  gradients $\leftarrow$ TapeGradient(loss, model.trainable_variables)
  gradients are computed using the TensorFlow function tf.GradientTape to calculate the gradient of the loss with respect to the trainable variables of the model.
  return gradients
end function

Algorithm 3 Noisy gradient descent with DP

function NoisyGradientDescent($\epsilon, \delta, \Delta, T$)
  Initialize model weights: $\theta \leftarrow$ RandomInitialization()
  noisy_count $\leftarrow$ LaplaceMechanism(sample_count, 1, ϵ)
  for $t = 1$ to T **do**
    grad_sum $\leftarrow$ [zeros_like(var) for var in model_trainable_variables]
    for (x_i, y_i) in zip(X, y) **do**
      clipped_x_i $\leftarrow$ L2_Clipping(x_i, Δ)
      grad_sum += Gradient(model, clipped_x_i, y_i)
    end for
    noisy_grad_sum $\leftarrow$ GaussianMechanism(grad_sum, Δ, ϵ, δ)
    noisy_avg_grad $\leftarrow$ [$\frac{g}{\text{noisy\_count}}$ for g in noisy_grad_sum]
    for j, grad in enumerate(noisy_avg_grad) **do**
      noisy_avg_grad[j] $\leftarrow$ Cast(grad, dtype = model.trainable_variables[j].dtype)
      model.trainable_variables[j] $-=$ noisy_avg_grad[j]
    end for
    grad_zip $\leftarrow$ zip(noisy_avg_grad, model.trainable_variables)
    model_optimizer.apply_gradients(grad_zip)
  end for
end function

Algorithm 4 Adding Differential Privacy to Trained Model

Set Privacy Parameters:
Set sensitivity (Δ), delta (δ), number of classes (C), and iterations (T)
Train Model with Differential Privacy:
Iterate over a range of epsilon (ϵ) values representing privacy levels
For each ϵ, train the model with DP using the function NoisyGradientDescent defined in Algorithm 3.
Evaluate Model Performance:
After training with each epsilon, evaluate the model on the test data and observe the performance score.

Noisy gradient descent algorithm. The core of DP-RFECV-FNN is the *NoisyGradientDescent* function, which encapsulates the DP mechanism using noisy gradient descent. In each iteration, gradients are computed, L2 Clipping is applied to enhance privacy, and then noisy gradients are calculated. The function *L2_Clipping* is used within the *NoisyGradientDescent* function to enforce L2 norm clipping on individual data points during the training of a model with DP. The model weights are updated using these noisy gradients, ensuring that the training process incorporates privacy-preserving measures. This function is stated in Algorithm 3.

Training and evaluation. The final loop is iterated over different epsilon values, representing various levels of privacy. For each epsilon, the model is trained using the noisy gradient descent function, and the performance is evaluated on the test set. Algorithm 4 outlines the procedure for adding DP in the trained model.

Integration of zero trust security in DP-RFECV-FNN. According to the principles of the zero trust model, all applications have to go through a verification step before getting access to the system's resources. After determining whether an application is benign or malicious, we integrated this step. The malicious applications will be rejected and discarded for further verification. Only the benign applications will be investigated further to check for verification and authentication to gain access. The least privileged access will be provided only to the verified APKs. Algorithm 5 states the integration of zero trust security with the proposed framework.

5. Result analysis

5.1. Experimental setup

The experiments are conducted on a computer with an Intel Core i9-7920X CPU @ 2.90 GHz, featuring 12 cores and 24 logical processors. 64-bit Linux Operating System is utilized. We use Python Programming Language for the implementation.

5.2. Performance metrics

To evaluate the performance of DP-RFECV-FNN, in different privacy scenarios while integrating DP mechanisms and the neural network model, the following evaluation metrics are used: Accuracy, Recall, Precision, F1-Score, True Positive Rate (TPR), and False Positive Rate (FPR). These metrics are represented in Eqs. (5)–(10):

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5}$$

Table 7
Hyperparameter settings for different models.

	LSTM	CNN	CapsNet	DP-RFECV-FNN
Layers & Architecture	Hidden Layers: 2 Neurons per Hidden Layer: (128, 128)	Configuration: 4 layers (2 Conv1D with 64 filters, kernel size 3; 2 MaxPooling1D, pool size 2), followed by 2 Dense layers (128 neurons, output 1 neuron).	Configuration: 1 Input layer, 2 Conv1D layers (30 filters, kernel size 3, stride 3), followed by a Capsule layer with digit capsule calculation.	Hidden Layers: 2 Neurons per Hidden Layer: (64, 32)
Activation functions	ReLu & Softmax	ReLu & Softmax	ReLu & Softmax	ReLu & Softmax
Optimizer	Adam	Adam	Adam	Adam
Loss function	categorical_crossentropy	categorical_crossentropy	categorical_crossentropy	categorical_crossentropy
Learning rate	0.001	0.001	0.001	0.001
Batch size	64	64	64	64
Epochs	100	100	100	100

Algorithm 5 Integration of zero trust security in malware detection.

Input: Dataset of APK files
Output: Access decision for each APK file
Classify Applications:
for each APK file in the dataset **do**
    Classify the APK as benign or malicious using the trained model
    if APK is classified as malicious **then**
        Reject the APK and prevent access
        Continue
    end if
    Zero Trust Verification:
    Perform authentication and verify the identity of the benign APK
    If APK fails verification, reject and prevent access
    If APK passes verification, grant access with the least privilege
    Logging:
    Log the classification and verification results
end for

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{8}$$

$$TPR = \frac{\sum_{i=1}^{m} [a(x_i) = +1][y_i = +1]}{\sum_{i=1}^{m} [y_i = +1]} \tag{9}$$

$$FPR = \frac{\sum_{i=1}^{m} [a(x_i) = +1][y_i = -1]}{\sum_{i=1}^{m} [y_i = -1]} \tag{10}$$

5.3. Detecting Android malware

We compared the performance of DP-RFECV-FNN with different baseline models, including the Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), and Capsule Network (CapsNet) and hyperparameters settings of each model is shown in Table 7. An extensive comparative analysis of malware detection accuracy achieved by DP-RFECV-FNN against these different baseline models are presented in Table 8 using both static and dynamic features. The performance of these models is measured under different privacy budgets, denoted by ϵ with values of 0.1, 0.5, and 1.0. Notably, DP-RFECV-FNN consistently outperforms the baseline models across all privacy settings, demonstrating the proposed framework's robustness and efficacy in achieving high accuracy in a privacy-preserving manner.

In the experiment involving static features, DP-RFECV-FNN exhibited remarkable accuracy, achieving up to 99.72% without privacy preservation. It maintained a high accuracy of up to 97.84% even when a privacy parameter ($\epsilon = 0.5$) was introduced. When considering dynamic features, DP-RFECV-FNN achieved an accuracy of up to 95.77% without privacy preservation and a slightly lower accuracy of 94.43% when $\epsilon = 0.5$. The comparative analysis depicted in Fig. 5 illustrates the F1-Score performance of the proposed framework against baseline models under various privacy settings, encompassing both static and dynamic features. DP-RFECV-FNN consistently outperformed the three base models in both scenarios.

We then conducted an extensive evaluation of the proposed model across different privacy parameter settings, ranging from $\epsilon = 0.1$ to $\epsilon = 10$, as depicted in Fig. 6. The figure indicates that, generally, an increase in the ϵ value corresponds to an enhancement in performance metrics. It is important to note that when the ϵ value exceeds 10, it represents a scenario without privacy protection. Particularly noteworthy is the fact that DP-RFECV-FNN achieved the highest recall in both static and dynamic analyses at $\epsilon = 0.1$, reflecting stringent privacy considerations. A model with the highest recall is adept at minimizing false negatives, ensuring the correct detection of a large number of actual positive cases.

Table 9 shows the evaluation of model performance in relation to resource consumption which is a vital consideration for IoT devices with limited resources. This comprehensive comparison demonstrates not only the accuracy of each model under static and dynamic analyses but also quantifies their training times and memory usage. The DP-RFECV-FNN shows a superior balance between high accuracy and low resource consumption compared with other baseline models. These compelling findings contribute significantly to the intersection of ML and privacy, underscoring the practical viability of DP-RFECV-FNN in real-world applications. This is particularly relevant in scenarios where maintaining high accuracy is crucial alongside the imposition of strict privacy requirements.

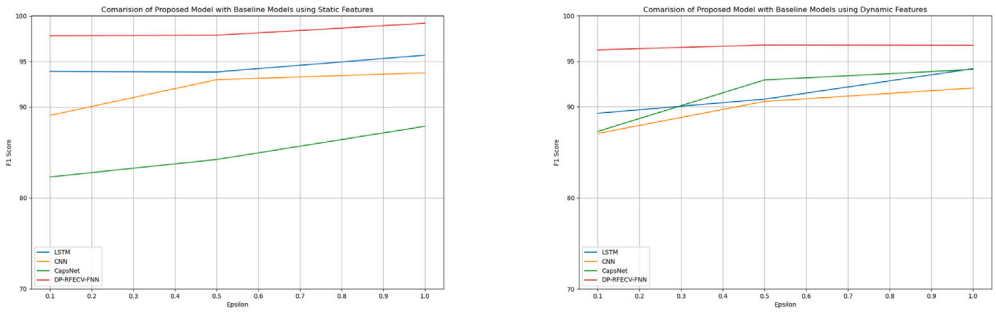
5.4. Classifying Android malware

We explored the performance of DP-RFECV-FNN in classifying malware categories using both static and dynamic features incorporating DP mechanisms against the baseline models, including LSTM, CNN, and CapsNet. Considering distinct privacy preservation levels denoted by ϵ set to 0.1, 0.5, and 1.0, the accuracy achieved in each privacy level was recorded in Table 10. Remarkably, DP-RFECV-FNN consistently exhibits superior accuracy across various scenarios. It achieved robust performance without privacy preservation, achieving an accuracy of 93.33%, surpassing the accuracy of LSTM, CNN, and CapsNet. Even under heightened privacy constraints, $\epsilon = 0.5$, DP-RFECV-FNN maintains a comparable accuracy of 92.09% when considering static features and accuracy of 89.54% when considering dynamic features that outperform the baseline models.

To the best of our knowledge, this is the first work on designing privacy-preserving neural networks in Android malware type classification using dynamic features. Fig. 7 shows the comparison of the F1-Score of the proposed model against the baseline models under

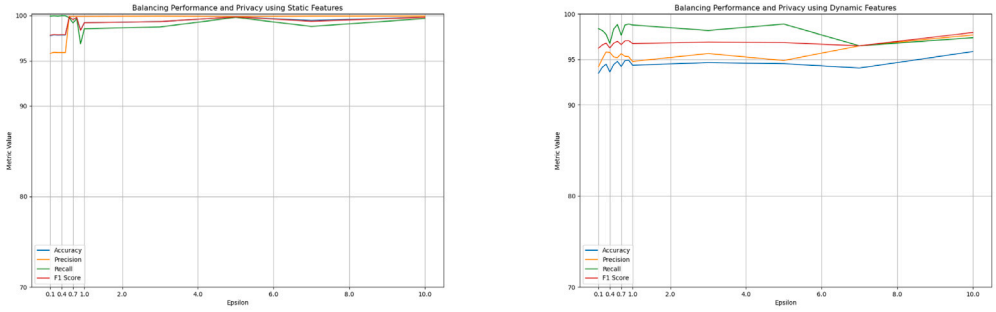
Table 8
Accuracy of DP-RFECV-FNN vs. baseline models for malware detection.

Models	Static analysis				Dynamic analysis			
	No DP	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1.0$	No DP	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1.0$
LSTM	97.69%	93.63%	93.65%	93.66%	92.32%	82.37%	84.69%	89.74%
CNN	93.48%	89.46%	92.76%	93.62%	93.01%	79.65%	85%	87.71%
CapsNet	94.77%	78.65%	81.27%	86.22%	90.73%	78.96%	87.84%	89.65%
DP-RFECV-FNN	**99.72%**	**97.78%**	**97.84%**	**99.21%**	**95.77%**	**93.49%**	**94.43%**	**94.36%**



(a) Using Static Features (b) Using Dynamic Features

Fig. 5. Trade-offs between privacy and utility, performance comparison using F1 Score of DP-RFECV-FNN with other baseline models in malware detection.



(a) Exploring Epsilon values in Android malware detection using static features (b) Exploring Epsilon values in Android malware detection using dynamic features

Fig. 6. Trade-offs in Android malware detection: Balancing performance and privacy in DP-RFECV-FNN.

Table 9
Comparison of resource consumption of DP-RFECV-FNN with other baseline models.

Model	Accuracy (%)		Training time (s)		Memory usage (MB)	
	Static analysis	Dynamic analysis	Static analysis	Dynamic analysis	Static analysis	Dynamic analysis
LSTM	97.69	92.32	3.46	2.72	20.72	20.78
CNN	93.48	93.01	3.25	3.63	15.22	12.02
CapsNet	94.77	90.73	3.53	3.05	15.73	7.81
DP-RFECV-FNN	**99.72**	**95.77**	**3.18**	**2.29**	**7.31**	**3.94**

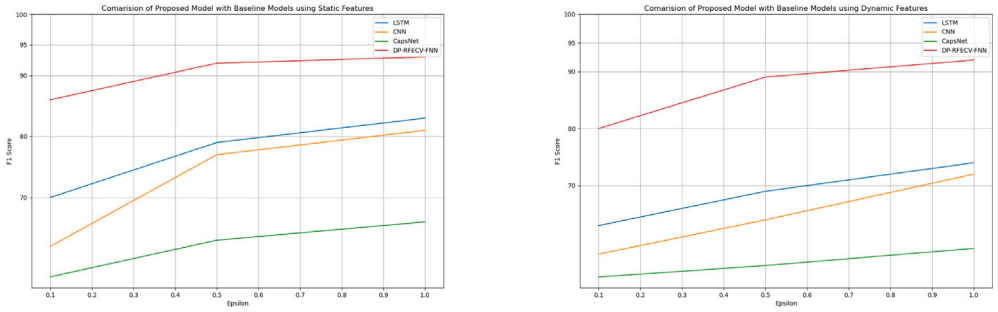
different privacy measures in classifying malware categories using both static and dynamic features. Fig. 7 demonstrates that the proposed model outperformed the base models in both cases.

We then assessed the performance of DP-RFECV-FNN under different privacy scenarios starting from $\epsilon = 0.1$ to $\epsilon = 10$ using both static and dynamic features and observed the trade-offs between privacy and utility. Fig. 8 shows that in both cases, DP-RFECV-FNN achieved a remarkable accuracy at $\epsilon = 0.4$, successfully preserving strict privacy considerations. These findings emphasize the efficacy of DP-RFECV-FNN in malware category classification and demonstrate stability to privacy-preserving measures while delivering significant accuracy.

We present the detailed performance analysis of DP-RFECV-FNN, for Android malware classification based on static features in Table 11 and dynamic features in Table 12 while considering the trade-off between privacy and utility. These tables demonstrate the effectiveness of the proposed framework across various malware categories, namely Adware, SMS, Ransomware, Riskware, and Trojan, while considering static features and Adware, Banker, SMS, and Riskware while considering dynamic features under different privacy preservation levels denoted by ϵ set to 0.2, 0.5, and 1.0 as well as without considering privacy preservation. DP-RFECV-FNN consistently exhibits strong recall, precision, and F1 scores across all the malware categories, even in the case of strict privacy preservation ($\epsilon = 0.5$). The trade-off is evident in the slight reduction of performance metrics with strengthened privacy that emphasizes the balance between preserving user privacy and maintaining accurate malware classification. The ability of DP-RFECV-FNN to maintain high utility while considering both static and dynamic features makes it a promising solution for Android malware classification with consideration of preserving the privacy of the user data.

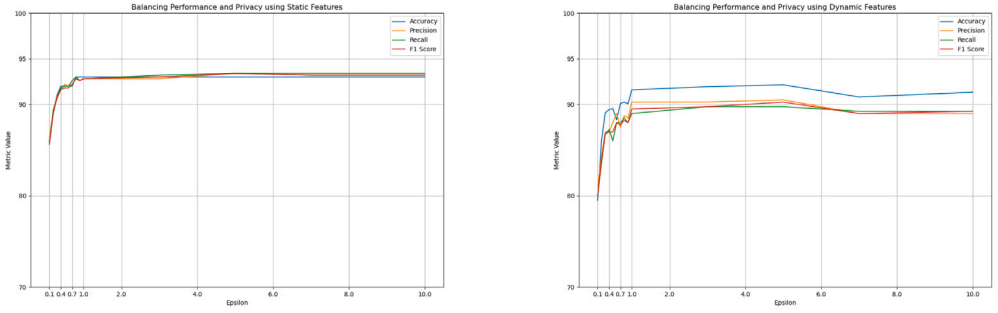
Table 10
Accuracy of DP-RFECV-FNN vs. baseline models for malware classification.

Models	Static analysis				Dynamic analysis			
	No DP	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1.0$	No DP	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1.0$
LSTM	93.02%	70.38%	79.32%	82.59%	92.81%	63.38%	69.96%	73.74%
CNN	92.80%	61.94%	76.68%	80.45%	93.77%	58.54%	64.71%	70.88%
CapsNet	83.49%	56.37%	62.77%	66.31%	90.73%	54.46%	56.45%	59.30%
DP-RFECV-FNN	**93.33%**	**85.72%**	**92.09%**	**92.73%**	**94.24%**	**79.5%**	**89.54%**	**91.59%**



(a) Using Static Features (b) Using Dynamic Features

Fig. 7. Trade-offs between privacy and performance: Performance comparison (F1 Score) of DP-RFECV-FNN with other baseline models in malware classification.



(a) Exploring Epsilon values in Android malware classification using static features (b) Exploring Epsilon values in Android malware classification using dynamic features

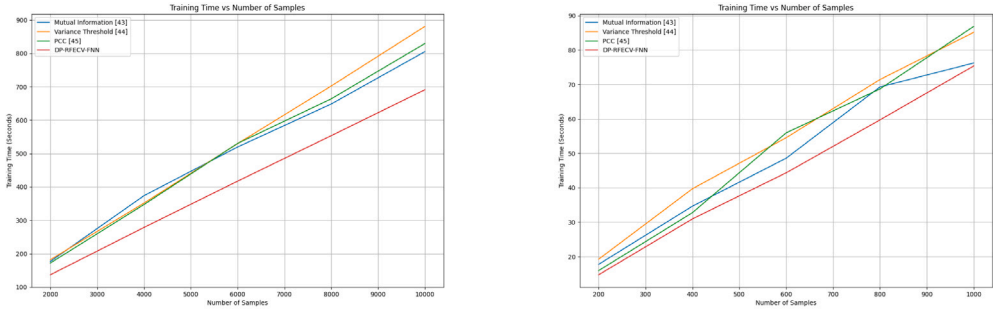
Fig. 8. Trade-offs in Android malware classification: Balancing performance and privacy in DP-RFECV-FNN.

5.5. Training time evaluation

The feature selection method employed in DP-RFECV-FNN is compared with other techniques found in the recent papers on Android malware detection regarding training time, as shown in Fig. 9. The comparison includes various feature selection methods such as Mutual Information [43], Variance Threshold [44], and Pearson Correlation Coefficient (PCC) [45] used in Android malware detection and we deployed these methods in our implementation setup and datasets. Fig. 9(a) presents the training duration for the CCCS-CIC-AndMal-2020 dataset, considering sample sizes from 2000 to 10 000. In contrast, Fig. 9(b) displays the training time for the CICMalDroid 2020 dataset with sample sizes ranging from 200 to 1000. DP-RFECV-FNN shows a significant reduction in training time across both datasets. This indicates that the DP-RFECV-FNN is more computationally efficient, which can be advantageous for large datasets or when computational resources are limited. To further explore the efficiency of the feature selection method used in DP-RFECV-FNN, Fig. 10 shows the comparative analysis of accuracy vs. number of features using the above two datasets. DP-RFECV-FNN outperforms others by achieving higher accuracy with a notably smaller set of features. It reaches an accuracy of 99.7% with only 19 features using the CCCS-CIC-AndMal-2020 dataset and achieves 95.8% accuracy with 33 features using the CICMalDroid 2020 dataset, whereas other methods, such as Mutual Information and PCC require a larger set of features to reach slightly lower accuracies. This substantial reduction in the number of features without a loss in accuracy emphasizes the potential of DP-RFECV-FNN for creating more interpretable models that are less complex and computationally more efficient.

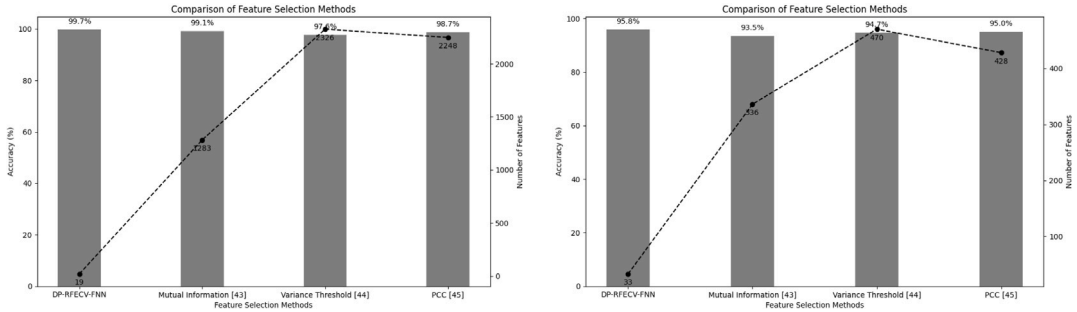
5.6. Testing data for zero trust

For testing the applications using DP-RFECV-FNN considering the zero trust framework, we conduct both static and dynamic analyses on each APK. For static analysis, we utilize DroidLysis [52], and for dynamic analysis, we employ the Mobile Security Framework (MobSF) [53]. Following the analysis of an APK, we assess whether it is benign or malicious. Based on this assessment, the APK is either granted access to the system or it will be blocked. Our approach does not check if an APK has been previously analyzed because, according to the zero trust principle, we must test every APK to finalize the authentication. We have tested on multiple APKs, and due to space constraints, we showed the result of two APKs in Tables 13 and 14. After selecting the APK, Droidlysis extracts static features such as permissions, activities, receivers, services, and providers. The main functionality of an APK is inferred from the Droidlysis report and by checking the main activity and the permissions it requests. For APK 1, the main activity listed is $com.bantu.trgame.WelcomeActivity$, which suggests that the app



(a) CCCS-CIC-AndMal-2020 Dataset (b) CICMalDroid 2020 Dataset

Fig. 9. Training time comparison with other feature selection methods.



(a) CCCS-CIC-AndMal-2020 Dataset (b) CICMalDroid 2020 Dataset

Fig. 10. Accuracy vs. Number of Features comparison with other feature selection methods.

Table 11
Trade-Off between privacy and utility by DP-RFECV-FNN in android malware classification using static features.

Proposed framework	Malware category	Performance metric	ϵ 0.2	0.5	1.0	No DP
DP-RFECV-FNN	Adware	Recall	0.82	0.90	0.91	0.90
		Precision	0.93	0.92	0.92	0.93
		F1 Score	0.87	0.91	0.92	0.91
		TPR	0.82	0.90	0.91	0.90
		FPR	0.02	0.02	0.02	0.01
	SMS	Recall	0.92	0.95	0.97	0.96
		Precision	0.88	0.90	0.90	0.93
		F1 Score	0.89	0.92	0.93	0.94
		TPR	0.92	0.95	0.97	0.96
		FPR	0.03	0.03	0.03	0.02
	Ransomware	Recall	0.96	0.97	0.98	0.98
		Precision	0.92	0.94	0.95	0.96
		F1 Score	0.94	0.95	0.97	0.97
		TPR	0.96	0.97	0.98	0.98
		FPR	0.02	0.02	0.01	0.009
	Riskware	Recall	0.91	0.92	0.93	0.92
		Precision	0.90	0.93	0.92	0.95
		F1 Score	0.90	0.92	0.92	0.93
		TPR	0.91	0.92	0.93	0.92
		FPR	0.02	0.02	0.02	0.01
	Trojan	Recall	0.86	0.86	0.85	0.91
		Precision	0.83	0.92	0.95	0.90
		F1 Score	0.85	0.89	0.90	0.90
		TPR	0.86	0.86	0.85	0.91
		FPR	0.04	0.02	0.02	0.02

is a game application. The set of permissions like INTERNET, ACCESS_NETWORK_STATE, and WRITE_EXTERNAL_STORAGE, are common for games that require to save data, retrieve game assets, or show

Table 12
Trade-Off between privacy and performance using dynamic features.

Proposed framework	Malware category	Performance metric	ϵ 0.2	0.5	1.0	No DP
DP-RFECV-FNN	Adware	Recall	0.77	0.70	0.76	0.92
		Precision	0.76	0.89	0.87	0.87
		F1 Score	0.77	0.79	0.81	0.89
		TPR	0.77	0.70	0.76	0.92
		FPR	0.03	0.01	0.01	0.01
	Banker	Recall	0.85	0.90	0.91	0.92
		Precision	0.77	0.79	0.85	0.91
		F1 Score	0.81	0.84	0.88	0.92
		TPR	0.85	0.90	0.91	0.92
		FPR	0.06	0.06	0.04	0.02
	SMS	Recall	0.95	0.96	0.97	0.98
		Precision	0.91	0.97	0.97	0.99
		F1 Score	0.93	0.97	0.97	0.99
		TPR	0.95	0.96	0.97	0.98
		FPR	0.06	0.02	0.02	0.007
	Riskware	Recall	0.77	0.90	0.92	0.91
		Precision	0.93	0.87	0.92	0.94
		F1 Score	0.84	0.88	0.92	0.93
		TPR	0.77	0.90	0.92	0.91
		FPR	0.02	0.04	0.03	0.02

ads. However, READ_PHONE_STATE, CHANGE_NETWORK_STATE, REORDER_TASKS, MOUNT_UNMOUNT_FILESYSTEMS are some of the dangerous permissions used by APK 1, which are out of its functionality. After that, DP adds random noise to this data to protect it from data-poisoning attacks by adversaries. Finally, the perturbed data is fed into the model of DP-RFECV-FNN, and APK 1 is categorized as malware. If an APK is categorized as malware from the static analysis, we do not proceed with dynamic analysis and block this APK from getting access to the system. For APK 2, the main activity

Table 13
Testing APK 1 for zero trust.

Sha256	00a3ecdc419616323cc2eee6d43fb65a9c058ede5abae9dab2a84c892974cd72	APK 1
Activities	com.bantu.trgame.WelcomeActivity,com.bantu.trgame.TRGame,com.facebook.FacebookActivity,com.facebook.CustomTabActivity,com.excelliance.assetsonly.base.ForwardActivity,com.excelliance.open.NextChapter,com.excelliance.open.AssistActivity,com.excelliance.open.PromptActivity,com.google.android.gms.auth.api.signin.internal.SignInHubActivity,com.android.billingclient.api.ProxyBillingActivity,com.google.android.gms.common.api.GoogleApiActivity,com.google.android.gms.ads.AdActivity,com.facebook.CustomTabMainActivity	
Permissions	INTERNET, WRITE_EXTERNAL_STORAGE, ACCESS_WIFI_STATE, ACCESS_NETWORK_STATE, READ_PHONE_STATE, GET_TASKS, READ_LOGS, SYSTEM_ALERT_WINDOW, CHANGE_NETWORK_STATE, MOUNT_UNMOUNT_FILESYSTEMS, READ_EXTERNAL_STORAGE, WAKE_LOCK, FOREGROUND_SERVICE, GET_DETAILED_TASKS, REORDER_TASKS, com.android.vending.BILLING, com.google.android.c2dm.permission.RECEIVE	
Providers	com.excelliance.assetsonly.preferences.provider.LBContentProvider,com.google.firebase.provider.FirebaseInitProvider,com.facebook.internal.FacebookInitProvider	
Receivers	com.excellence.assetsonly.base.BGReceiver,com.google.firebase.iid.FirebaseInstanceIdReceiver,com.google.android.gms.measurement.AppMeasurementReceiver,com.google.android.gms.measurement.AppMeasurementInstallReferrerReceiver,com.facebook.CurrentAccessTokenExpirationBroadcastReceiver	
Services	com.excelliance.assetsonly.debug.LBSdkCrashReportService,com.excelliance.assetsonly.base.AssistService,com.excelliance.assetsonly.base.BaseService,com.excelliance.assetsonly.main.BGService,com.google.android.gms.auth.api.signin.RevocationBoundServic,com.google.firebase.components.ComponentDiscoveryService,com.google.firebase.iid.FirebaseInstanceIdService,com.google.android.gms.measurement.AppMeasurementService,com.google.android.gms.measurement.AppMeasurementJobService	
Main activity	com.bantu.trgame.WelcomeActivity	
Main functionality	Game App	
Excessive permissions	READ_PHONE_STATE, CHANGE_NETWORK_STATE, REORDER_TASKS, MOUNT_UNMOUNT_FILESYSTEMS	
Benign or Malware?	Malware	

Table 14
Testing APK 2 for zero trust.

Sha256	004ab23cec034b0aad323a963847d9b6cefde24c23cef9389f2c62f49650928b	APK 2
Activities	com.radinelqaa.myphone.findphone.view.activities.RateUsAct,com.radinelqaa.myphone.findphone.More_Apps,com.google.android.gms.ads.AdActivity,com.radinelqaa.myphone.findphone.view.activities.SplashScreen,com.radinelqaa.myphone.findphone.view.activities.DialogeActivity,com.radinelqaa.myphone.findphone.MainActivity,com.radinelqaa.myphone.findphone.view.activities.ClapToFindPhone_ACT,com.radinelqaa.myphone.findphone.view.activities.StopBuzz	
Permissions	CAMERA, FLASHLIGHT, INTERNET, RECORD_AUDIO, FOREGROUND_SERVICE, VIBRATE, READ_EXTERNAL_STORAGE, ACCESS_NETWORK_STATE, WAKE_LOCK	
Providers	com.google.android.gms.ads.MobileAdsInitProvider	
Receivers	com.google.android.gms.measurement.AppMeasurementReceiver,com.google.android.gms.measurement.AppMeasurementInstallReferrerReceiver	
Services	com.radinelqaa.myphone.findphone.model.service.VoiceDetectService,com.radinelqaa.myphone.findphone.model.service.SoundPlayService,com.radinelqaa.myphone.findphone.model.service.service,com.google.android.gms.measurement.AppMeasurementService,com.google.android.gms.measurement.AppMeasurementJobService	
Main activity	com.radinelqaa.myphone.findphone.view.activities.SplashScreen	
Main functionality	Phone Finder App	
Excessive permissions	RECORD_AUDIO, READ_EXTERNAL_STORAGE	
Least privileged permissions	CAMERA, FLASHLIGHT, INTERNET, FOREGROUND_SERVICE, VIBRATE, ACCESS_NETWORK_STATE, WAKE_LOCK	
Benign or Malware?	Benign	

is *com.radinelqaa.myphone.findphone.view.activities.SplashScreen* was found in the DroidLysis report. It is an app that helps to find the phone. It wants to use the camera and flashlight, which helps to locate the phone in the dark or take a picture of where it is. Like APK 1, static features of APK 2 are extracted using DroidLysis. After adding noise by DP to make the data anonymous, it is fed into the model of DP-RFECV-FNN, and the model categorizes APK 2 as a benign app. After further analysis, we found that despite being a benign app, there are some excessive permissions such as READ_PHONE_STATE, CHANGE_NETWORK_STATE, REORDER_TASKS, MOUNT_UNMOUNT_FILESYSTEMS used by APK 2. The least privileged permissions will be refined to exclude any excessive permissions during system access. Before providing this APK 2 access to the system, we do dynamic analysis also on it using MobSF. MobSF uses an emulator where the app is installed, and the run-time behavior of the app is examined, such as logs of all system calls and network traffic, and records if there is any malicious activity found. APK 2 is categorized as benign based on the dynamic analysis report, and finally, APK 2 will get access to the system with the least privileged access rights.

Granting the least privileged access is determined through the process of static and dynamic analysis of applications using tools like DroidLysis and MobSF, respectively. The static analysis via DroidLysis examines the manifest and code for declared permissions and activities to determine its intended use. The dynamic analysis by MobSF observes the app in execution, monitoring for runtime permissions and actual behavior. If the permissions requested by the application are found to be excessive for its stated purpose, they are flagged. These flagged permissions will be blocked while accessing the system resources, and only those permissions related to its functionalities will be granted. The common permissions between the declared permissions and expected permissions based on functionality are listed, which will be considered for the least privileged access rights shown in Table 14. The least privilege is ensured by allowing only the minimum access necessary for the application to function as intended.

To ensure that applications maintain proper operation following the modification of the access settings, our system incorporates a testing mechanism. Initially, applications are analyzed using static and dynamic methods to determine the minimal set of permissions they require to function as intended. After identifying these least privileged access settings, our system utilizes a monitoring strategy to observe the application's behavior in real-time. If any issue is detected during this phase, trigger alerts that will require the controlled modification of the access settings.

6. Discussion and future work

The evaluation of the DP-RFECV-FNN framework utilized the CCCS-CIC-AndMal-2020 and CICMalDroid 2020 datasets, which may not entirely encapsulate the vast diversity of Android malware that exists in real-world scenarios that could affect the generalization of the proposed solution across all malware types and scenarios. However, the DP-RFECV-FNN detects new malware by incorporating a dynamic analysis approach. Dynamic analysis also helps to counter evasion attacks. Dynamic analysis runs the application in a controlled environment (Sandbox) to observe its behavior. This approach can identify malicious activities that occur during execution. It analyzes the run-time behavior of the applications, such as network traffic, run-time permissions, system calls. By observing the real-time behavior of the applications, DP-RFECV-FNN identifies patterns and anomalies that static analysis cannot perform and detects malicious activities. This dynamic feature extraction plays a significant role in recognizing new malware or countering evasion attacks, which can be bypassed by traditional detection methods. Additionally, by incorporating DP, we add an additional layer of security, DP-RFECV-FNN prevents attackers from inferring sensitive information from the output of the model to protect against data exploitation and enhances the security of the detection process. Our approach also relies on a zero trust model, which assumes that no application is safe without thorough verification, which further strengthens the defense mechanisms against novel malware attacks.

In the future, we will extend this research by incorporating diverse datasets containing recent malware. In this article, we experimented with only one approach to privacy preservation, namely Differential Privacy. In the future, we will integrate other privacy-preservation techniques such as Federated Learning, Homomorphic Encryption, and Secure Multi-party Computation and make a comparative analysis. By systematically assessing their performance, computational overhead, and adaptability to different scenarios, we intend to offer insights that will guide researchers in choosing the most suitable privacy-preservation technique based on their specific requirements and constraints.

7. Conclusion

As Android plays a key component in device operations within the expansive IoT framework, particularly in healthcare devices, securing applications is compulsory for protecting user-sensitive data. The detection and analysis of Android malware are thus essential measures for preserving confidential information and ensuring the robustness of IoT networks. In this article, we proposed DP-RFECV-FNN for securing IoT networks against Android malware attacks. DP-RFECV-FNN incorporated the principles of the zero trust model and the strengths of the DP mechanisms together with an FNN. DP-RFECV-FNN ensures that no applications will be trusted automatically following the zero trust model. Every application is tested, and only the benign applications are verified and authenticated further to gain access to the system. DP-RFECV-FNN achieves remarkable accuracy and a balance between performance and user privacy. The experimental results show superior performance metrics, exceeding the existing state-of-the-art solutions in terms of accuracy while ensuring robust privacy guarantees. The achieved accuracy, ranging from 97.78% to 99.21% for static features and 93.49% to 94.36% for dynamic features while detecting benign or malicious applications under various privacy budgets ($\epsilon = 0.1$ to $\epsilon = 1.0$) which emphasizes the effectiveness of DP-RFECV-FNN. Furthermore, The DP-RFECV-FNN model remarkably balances high detection accuracy with efficient use of resources which is proved by lower memory consumption. In addition to the higher accuracy in malware detection, the DP-RFECV-FNN model also stands out for the training speed. The faster training process, compared to traditional approaches is crucial for maintaining operational efficiency in IoT networks. This research significantly contributes to the field of IoT networks in improving zero trust security and offers a reliable and privacy-preserving solution to prevent the existing and new threats posed by Android malware in interconnected IoT networks.

CRediT authorship contribution statement

Faria Nawshin: Writing – original draft, Visualization, Validation, Methodology, Investigation, Data curation, Conceptualization. **Devrim Unal:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Funding acquisition, Formal analysis, Conceptualization. **Mohammad Hammoudeh:** Writing – review & editing, Validation, Methodology. **Ponnuthurai N. Suganthan:** Writing – review & editing, Supervision, Methodology, Investigation, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We used public datasets which are cited in the paper.

Acknowledgment

Open Access funding provided by the Qatar National Library.

References

[1] Z. Ren, H. Wu, Q. Ning, I. Hussain, B. Chen, End-to-end malware detection for android IoT devices using deep learning, Ad Hoc Netw. 101 (2020) 102098.
[2] D. Unal, S. Bennbaia, F.O. Catak, Machine learning for the security of healthcare systems based on Internet of Things and edge computing, in: Cybersecurity and Cognitive Science, Elsevier, 2022, pp. 299–320.
[3] G. Zhang, Y. Li, X. Bao, C. Chakarborty, J.J. Rodrigues, L. Zheng, X. Zhang, L. Qi, M.R. Khosravi, TSDroid: A novel Android malware detection framework based on temporal & spatial metrics in IoMT, ACM Trans. Sensor Netw. 19 (3) (2023) 1–23.
[4] H.M. Alshahrani, Droid-iot: Detect android iot malicious applications using ml and blockchain, Comput. Mater. Contin. 70 (1) (2021) 739–766.
[5] R. Kumar, X. Zhang, R.U. Khan, A. Sharif, Research on data mining of permission-induced risk for android IoT devices, Appl. Sci. 9 (2) (2019) 277.
[6] Stfalcon LLC, Internet of medical things security, 2023, Accessed 31-01-2024. URL https://www.linkedin.com/pulse/internet-medical-things-security-stfalconcom-glkff/.
[7] M. Amin, D. Shehwar, A. Ullah, T. Guarda, T.A. Tanveer, S. Anwar, A deep learning system for health care IoT and smartphone malware detection, Neural Comput. Appl. (2020) 1–12.
[8] Z. Ji, Z.C. Lipton, C. Elkan, Differential privacy and machine learning: a survey and review, 2014, arXiv preprint arXiv:1412.7584.
[9] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, et al., Privacy-preserving machine learning with fully homomorphic encryption for deep neural network, IEEE Access 10 (2022) 30039–30054.
[10] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, L. van der Maaten, Crypten: Secure multi-party computation meets machine learning, Adv. Neural Inf. Process. Syst. 34 (2021) 4961–4973.
[11] X. Deng, H. Tang, X. Pei, D. Li, K. Xue, MDHE: A malware detection system based on trust hybrid user-edge evaluation in IoT network, IEEE Trans. Inf. Forensics Secur. (2023).

[12] W. Huang, X. Xie, Z. Wang, J. Feng, G. Han, W. Zhang, ZT-Access: A combining zero trust access control with attribute-based encryption scheme against compromised devices in power IoT environments, Ad Hoc Netw. 145 (2023) 103161.
[13] H. Fereidooni, M. Conti, D. Yao, A. Sperduti, ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications, in: 2016 8th IFIP International Conference on New Technologies, Mobility and Security, NTMS, IEEE, 2016, pp. 1–5.
[14] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, S. Son, De-LADY: Deep learning based Android malware detection using Dynamic features, J. Internet Serv. Inf. Secur. 11 (2) (2021) 34–45.
[15] C. Dwork, Differential privacy, in: International Colloquium on Automata, Languages, and Programming, Springer, 2006, pp. 1–12.
[16] M. Abadi, A. Chu, I. Goodfellow, H.B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 308–318.
[17] L. Sun, S. Bao, S. Ci, X. Zheng, L. Guo, Y. Luo, Differential privacy-preserving density peaks clustering based on shared near neighbors similarity, IEEE Access 7 (2019) 89427–89440.
[18] F. Liu, Generalized gaussian mechanism for differential privacy, IEEE Trans. Knowl. Data Eng. 31 (4) (2018) 747–756.
[19] G. Bendiab, S. Shiaeles, A. Alruban, N. Kolokotronis, IoT malware network traffic classification using visual representation and deep learning, in: 2020 6th IEEE Conference on Network Softwarization, NetSoft, IEEE, 2020, pp. 444–449.
[20] M. Shobana, S. Poonkuzhali, A novel approach to detect IoT malware by system calls using Deep learning techniques, in: 2020 International Conference on Innovative Trends in Information Technology, ICITIIT, IEEE, 2020, pp. 1–5.
[21] S. Ali, O. Abusabha, F. Ali, M. Imran, T. ABUHMED, Effective multitask deep learning for iot malware detection and identification using behavioral traffic analysis, IEEE Trans. Netw. Serv. Manag. (2022).
[22] R. Chaganti, V. Ravi, T.D. Pham, Deep learning based cross architecture internet of things malware detection and classification, Comput. Secur. 120 (2022) 102779.
[23] T. Lu, Y. Du, L. Ouyang, Q. Chen, X. Wang, Android malware detection based on a hybrid deep learning model, Secur. Commun. Netw. 2020 (2020) 1–11.
[24] N. Zhang, Y.-a. Tan, C. Yang, Y. Li, Deep learning feature exploration for android malware detection, Appl. Soft Comput. 102 (2021) 107069.
[25] J. Kim, Y. Ban, E. Ko, H. Cho, J.H. Yi, MAPAS: a practical deep learning-based android malware detection system, Int. J. Inf. Secur. 21 (4) (2022) 725–738.
[26] O.N. Elayan, A.M. Mustafa, Android malware detection using deep learning, Procedia Comput. Sci. 184 (2021) 847–852.
[27] A. Albakri, F. Alhayan, N. Alturki, S. Ahamed, S. Shamsudheen, Metaheuristics with deep learning model for cybersecurity and Android malware detection and classification, Appl. Sci. 13 (4) (2023) 2172.
[28] R. Gálvez, V. Moonsamy, C. Diaz, Less is More: A privacy-respecting Android malware classifier using federated learning, 2020, arXiv preprint arXiv:2007.08319.
[29] C. Jiang, K. Yin, C. Xia, W. Huang, FedHGCDroid: An adaptive multi-dimensional federated learning for privacy-preserving android Malware classification, Entropy 24 (7) (2022) 919.
[30] A. Mahindru, H. Arora, Dnndroid: Android malware detection framework based on federated learning and edge computing, in: International Conference on Advancements in Smart Computing and Information Security, Springer, 2022, pp. 96–107.
[31] R. Sun, X. Yuan, P. He, Q. Zhu, A. Chen, A. Gregio, D. Oliveira, X. Li, Learning fast and slow: Propedeutica for real-time malware detection, IEEE Trans. Neural Netw. Learn. Syst. 33 (6) (2021) 2518–2529.
[32] Y. Chai, L. Du, J. Qiu, L. Yin, Z. Tian, Dynamic prototype network based on sample adaptation for few-shot malware detection, IEEE Trans. Knowl. Data Eng. 35 (5) (2022) 4754–4766.
[33] P. Bhat, S. Behal, K. Dutta, A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning, Comput. Secur. 130 (2023) 103277.
[34] C. Li, Q. Lv, N. Li, Y. Wang, D. Sun, Y. Qiao, A novel deep framework for dynamic malware detection based on API sequence intrinsic features, Comput. Secur. 116 (2022) 102686.
[35] A.T. Kabakus, DroidMalwareDetector: A novel Android malware detection framework based on convolutional neural network, Expert Syst. Appl. 206 (2022) 117833.
[36] A. Mahindru, A. Sangal, SOMDROID: Android malware detection by artificial neural network trained using unsupervised learning, Evol. Intell. 15 (1) (2022) 407–437.
[37] E.B. Fernandez, A. Brazhuk, A critical analysis of Zero Trust Architecture (ZTA), Comput. Stand. Interfaces 89 (2024) 103832.
[38] D.S. Keyes, B. Li, G. Kaur, A.H. Lashkari, F. Gagnon, F. Massicotte, EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics, in: 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge, RDAAPS, IEEE, 2021, pp. 1–12.
[39] A. Rahali, A.H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, F. Massicotte, Didroid: Android malware classification and characterization using deep image learning, in: 2020 the 10th International Conference on Communication and Network Security, 2020, pp. 70–82.
[40] S. Mahdavifar, A.F.A. Kadir, R. Fatemi, D. Alhadidi, A.A. Ghorbani, Dynamic android malware category classification using semi-supervised deep learning, in: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech, IEEE, 2020, pp. 515–522.
[41] S. Mahdavifar, D. Alhadidi, A.A. Ghorbani, Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder, J. Netw. Syst. Manage. 30 (2022) 1–34.
[42] L. Cavallaro, CopperDroid: On the reconstruction of Android malware behaviors, in: HackInBo 2014, Royal Holloway University of London, London, 2014, Accessed: 03-01-2024. URL http://s2lab.isg.rhul.ac.uk/.
[43] C. Zhao, C. Wang, W. Zheng, Android malware detection based on sensitive permissions and apis, in: International Conference on Security and Privacy in New Computing Environments, Springer, 2019, pp. 105–113.
[44] S. Rawat, R. Phira, P. Natu, Use of machine learning algorithms for Android app malware detection, in: 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques, ICEECCOT, IEEE, 2021, pp. 448–454.
[45] L. Gong, Z. Li, H. Wang, H. Lin, X. Ma, Y. Liu, Overlay-based Android malware detection at market scales: Systematically adapting to the new technological landscape, IEEE Trans. Mob. Comput. 21 (12) (2021) 4488–4501.
[46] B.H. Menze, B.M. Kelm, R. Masuch, U. Himmelreich, P. Bachert, W. Petrich, F.A. Hamprecht, A comparison of random forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data, BMC Bioinform. 10 (2009) 1–16.
[47] Y. Liu, Y. Mu, K. Chen, Y. Li, J. Guo, Daily activity feature selection in smart homes based on pearson correlation coefficient, Neural Process. Lett. 51 (2020) 1771–1787.
[48] A.Z. Mustaqim, S. Adi, Y. Pristyanto, Y. Astuti, The effect of recursive feature elimination with cross-validation (RFECV) feature selection algorithm toward classifier performance on credit card fraud detection, in: 2021 International Conference on Artificial Intelligence and Computer Science Technology, ICAICST, IEEE, 2021, pp. 270–275.
[49] J. Sung, S. Han, H. Park, S. Hwang, S.J. Lee, J.W. Park, I. Youn, Classification of stroke severity using clinically relevant symmetric gait features based on recursive feature elimination with cross-validation, IEEE Access 10 (2022) 119437–119447.
[50] M. Kamaladevi, V. Venkataraman, K.R. Sekar, Tomek link undersampling with stacked ensemble classifier for imbalanced data classification, Ann. Rom. Soc. Cell Biol. (2021) 2182–2190.
[51] M.M. Ahsan, M.P. Mahmud, P.K. Saha, K.D. Gupta, Z. Siddique, Effect of data scaling methods on machine learning algorithms and model performance, Technologies 9 (3) (2021) 52.
[52] Cryptax, Droidlysis: a tool to analyze android applications, 2023, https://github.com/cryptax/droidlysis, Accessed 03-01-2024.
[53] MobSF, Mobile security framework (MobSF), 2023, https://github.com/MobSF/Mobile-Security-Framework-MobSF, Accessed 08-01-2024.

Faria Nawshin is a Graduate Assistant at the KINDI Center for Computing Research, College of Engineering, Qatar University. She is currently pursuing a Ph.D. degree in Computer Science at College of Engineering, Qatar University. She obtained her B.Sc degree in Computer Science & Engineering and M.Sc degree in Computer Science from American International University-Bangladesh (AIUB) in 2016 and 2017 respectively. Her research interests include federated learning, deep neural network, mobile computing and cyber security.

Devrim Unal, Ph.D., Senior Member, IEEE, is a Research Associate Professor of Cyber Security at the KINDI Center for Computing Research, College of Engineering, Qatar University. He obtained his M.Sc. degree in Telematics from Sheffield University, UK and Ph.D. degree in Computer Engineering from Bogazici University, Turkey in 1998 and 2011, respectively. Dr. Unal's research interests include cyber–physical systems and IoT security, secure and robust artificial intelligence and security of next generation networks.

Mohammad Hammoudeh is Saudi Aramco Cybersecurity Chair Professor with the Information & Computer Science Department, King Fahd University of Petroleum & Minerals, Saudi Arabia. His research interests include the applications of zero trust security to internet-connected critical national infrastructures and blockchain.

Ponnuthurai Nagaratnam Suganthan received the B.A degree and M.A degree in Electrical and Information Engineering from the University of Cambridge, UK in 1990, 1992 and 1994, respectively. He received an honorary doctorate (i.e. Doctor Honoris Causa) in 2020 from University of Maribor, Slovenia. After completing his Ph.D. research in 1995, he served as a pre-doctoral Research Assistant in the Dept of Electrical Engineering, University of Sydney in 1995–96 and a lecturer in the Dept of Computer Science and Electrical Engineering, University of Queensland in 1996–99. Since August 2022, he has been with KINDI Centre for Computing Research, Qatar University, as a research professor. He was an Editorial Board Member of the Evolutionary Computation Journal, MIT Press (2013–2018). He is/was an associate editor of the Applied Soft Computing (Elsevier, 2018-), Neurocomputing (Elsevier, 2018-), IEEE Trans on Cybernetics (2012–2018), IEEE Trans on Evolutionary Computation (2005–2021), Information Sciences (Elsevier) (2009 -), Pattern Recognition (Elsevier) (2001 -) and IEEE Trans on SMC: Systems (2020 -) Journals. He is a founding co-editor-in-chief of Swarm and Evolutionary Computation (2010 -), an SCI Indexed Elsevier Journal. His co-authored SaDE paper (published in April 2009) won the ''IEEE Trans. on Evolutionary Computation outstanding paper award'' in 2012. His research interests include randomization-based learning methods, swarm and evolutionary algorithms, pattern recognition, deep learning and applications of swarm, evolutionary & machine learning algorithms. He was selected as one of the highly cited researchers by Thomson Reuters Science Citation yearly from 2015 to 2022 in computer science. He served as the General Chair of the IEEE SSCI 2013. He has been a member of the IEEE (S'91, M'92, SM'00, Fellow 2015) since 1991 and an elected AdCom member of the IEEE Computational Intelligence Society (CIS) in 2014–2016. He was an IEEE CIS distinguished lecturer (DLP) in 2018–2021.