International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 25 Issue 01, JAN, 2025

Hybrid Machine Learning Models for the Detection and Prevention of Cyber Fraud Apps

Sk. Jabeerulla[1],
Asst. Professor,
Department Of Computer Science And Engineering,
Vignan's Lara Institute of Technology and Science,
Guntur, India,
jabeershaik1482@gmail.com

Madati Krishna Vamsi[2]
Department Of Computer Science And Engineering,
Vignan's Lara Institute of Technology and Science,
Guntur, India,
krishnavamsi.madati@gmail.com

Koritala Nagaraju[3]
Department Of Computer Science And Engineering,
Vignan's Lara Institute of Technology and Science,
Guntur, India,
koritalanagaraju8998@gmail.com

Govindu Krishna Sai[4]
Department Of Computer Science And Engineering,
Vignan's Lara Institute of Technology and Science,
Guntur, India,
krishnasaigovindu789@gmail.com

Nagam Guru Venkata Lokesh[5]
Department Of Computer Science And Engineering,
Vignan's Lara Institute of Technology and Science,
Guntur, India,
nagamlokesh93@gmail.com

***Abstract*—The rapid growth of Android applications has increased the risk of cyber fraud, because the malicious applications exploit the permissions of the users for carrying out unauthorized activities like data theft, premium SMS fraud, and ransomware attacks. The traditional methods of malware detection are signature- and heuristic-based approaches, which cannot cope with sophisticated attacks such as zero-day malware and obfuscation techniques. It will introduce a hybrid methodology, synergistically combining Support Vector Machines and Artificial Neural Networks, in order to identify malware by permission patterns. A Genetic Algorithm (GA) is used in the system for feature selection that optimizes the dataset for accuracy and reduces the computational overhead.**

The ANN model shows better detection performance with an accuracy of 94.2%, which is higher than that of SVM, which achieves 91.8%. The framework is further enhanced by the use of a real-time detection system as a Flask-based web application, where users can upload Android Package (APK) files for analysis. The application extracts permissions based on static analysis techniques and returns classification results as either malicious or benign with a confidence score.

Some key contributions of this research include ANN integration into the improvement of nonlinear feature modeling, GA usage to optimally choose features, and the creation of a real-time malware detection system that is scalable and user-friendly. The framework proposed hereby shows the power of machine learning in enhancing Android application-based cybersecurity and presents a stepping stone for further research work in malware detection.

I. Introduction

A. Background

Android is the leader in the mobile landscape, with billions of users and devices spread worldwide. Being an open-source platform, it encourages an active ecosystem of developers and applications. Yet, such openness leaves users exposed to enormous security risks, mainly from malicious applications distributed by third-party platforms and sometimes even official application stores like Google Play.

Malicious applications, or malware, exploit user permissions to execute unauthorized actions such as:

- **Data Theft:** Accessing sensitive user data such as contacts, messages, and browsing history.
- **Premium SMS Fraud:** Sending expensive SMS messages without user consent.
- **Ransomware Attacks:** Encrypting user files and demanding money for decryption.

The threat has exponentially increased cyber fraud. Global financial losses have crossed billions of dollars per year. With mobile applications now central to essential daily functions, including banking, communication, and healthcare, threats and risks amplify.

B. Problem Statement

The conventional approaches used for malware detection rely on either signature-based or heuristic-based methods. Signature-based methods are highly efficient for known threats but fail against zero-day attacks or malware employing obfuscation techniques. Heuristic-based methods are prone to high false-positive rates, often flagging harmless applications as malicious.

Traditional approaches have limitations; hence, more adaptive and scalable solutions are needed. Machine Learning (ML) models offer a promising alternative, but several challenges remain:

- **High-Dimensional Data:** Android applications request a large number of permissions, many of which are redundant or irrelevant for detecting malicious intent.
- **Model Scalability:** The ability of models to perform well on large datasets and varied types of malware.
- **Real-Time Applicability:** Deploying ML models in user-friendly, scalable systems for real-world use.

C. Objectives

This research is meant to be a hybrid machine. A learning model that detects and prevents cyber fraud in An- droid applications, meeting the following significant challenges:

- **Permissions-Based Detection:** Using static analysis to extract permissions from APK files as features for ML models.
- **Hybrid Model Integration:** Employing SVM for baseline performance and ANN for non-linear patterns.
- **Feature Optimization:** Using GA to reduce redundant permissions and improve computational efficiency.
- **Real-Time Deployment:** Designing a scalable, web-based detection system using Flask to enable real-time analysis.

D. Contributions

Key contributions of this work are:

- **ANN for Malware Detection:** Integration of ANN with permissions-based malware detection to achieve better accuracy and flexibility.
- **Feature Selection Using GA:** Demonstrates that GA can optimize large permission datasets, minimizing model complexity without losing performance.
- **Flask Real-Time Application:** A user-friendly system that makes machine learning predictions actionable and useful for end users.
- **Benchmarking Analysis:** The performance of SVM and ANN models is critically evaluated on Android malware datasets, thereby providing a comparative assessment of the strengths and weaknesses of both.

E. Significance

This paper overcomes significant challenges associated with detection of Android malware by presenting an integration of current state of the art approaches within machine learning and deployable solutions. The proposed system demonstrates the efficacy of combining static analysis with hybrid ML models to effectively detect and mitigate cyber threats, paving the way for future innovations in mobile cybersecurity.

II. Literature Survey

A. Static Analysis

Static analysis involves examining an application's code, manifest files, and permissions without executing the APK. The reasons why this approach is used widely in malware detection is because of its computational efficiency, where it can easily handle the large datasets. Tools like Androguard allow for permissions extraction to identify patterns indicative of malicious behavior.

Strengths of Static Analysis:

- Efficiency: Processes a large number of applications quickly.
- Scalability: Suitable for analyzing thousands of apps simultaneously.
- Reproducibility: Consistent feature extraction across analyses.

Limitations of Static Analysis:

- Code Obfuscation: Malware developers often hide their code to evade detection.
- Lack of Behavioral Insights: Does not capture runtime behavior like API calls or network activity.
- Limited Zero-Day Detection: Relies heavily on predefined patterns, which may not generalize to new threats.

B. Dynamic Analysis

Dynamic analysis evaluates an application's behavior during runtime in a sandbox environment. This approach identifies malicious activities such as network communication, file access, and API calls.

Advantages:

- Behavioral Insights: Captures real-time malicious activities.
- Zero-Day Detection: Identifies new malware based on suspicious behavior patterns.
- Comprehensive: Observes all operational contexts of an application.

Disadvantages:

- Resource-Intensive: Time- and computation-dependent.
- Scalability Issues: Not feasible for large-scale analysis.
- Evasion Techniques: Malware can detect sandbox environments and modify its behavior.

C. Machine Learning for Malware Detection

Machine Learning offers significant advantages over traditional methods by identifying complex data patterns. Features extracted from Android applications, such as permissions and system calls, serve as input for ML models.

1) Support Vector Machine (SVM): The robustness of SVM in handling high-dimensional data has established it as a go-to approach for detecting malware.

Limitations:

- Struggles with non-linear patterns without kernel functions.
- Computationally expensive for large datasets.

2) Artificial Neural Networks (ANNs): ANNs have great capabilities in modeling non-linear relationships and interactions between features, hence, they can handle large and complex datasets efficiently.

Limitations:

- Require hyperparameter tuning.
- Computationally intensive during training.

D. Feature Selection Methods

Feature selection enhances model performance by eliminating irrelevant features.

1) Genetic Algorithm (GA): This approach uses a natural selection-based strategy to identify an optimal subset of features, where their relevance is assessed by a fitness score reflecting the performance of the model.

Advantages:

- Handles high-dimensional feature spaces.
- Captures interactions between permissions.

III. Proposed Methodology

A. Framework Overview

This section outlines the comprehensive methodology adopted for the detection and prevention of cyber fraud targeting Android applications. The proposed framework encompasses static analysis, feature selection using Genetic Algorithm (GA), hybrid machine learning models, and real-time deployment. Each phase is explained using flowcharts.

1) Dataset Preparation: Collect and preprocess benign and malicious APK files.
2) Permissions Extraction: Extract static features (permissions) from APKs through manifest file analysis.
3) Feature Selection: Use GA for the optimization of the permission set by removing irrelevant and redundant features.
4) Model Training: SVM and ANN models are to be trained using the optimized feature set.
5) Real-Time Deployment: A Flask-based web application is to be implemented for real-time malware detection.

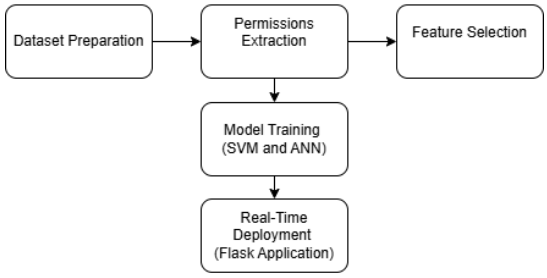

Fig. 1. Overall Framework

B. Dataset Preparation

The dataset for this study is categorized into two types of APK files: benign and malicious.

- **Benign Apps:** 1,500 APKs gathered from the period of 2015–2017.
- **Malicious Apps:** 1,200 APKs distributed across various malware categories:
  - Adware
  - Ransomware
  - SMS Malware
  - Scareware
  - Premium SMS Malware

All APK files were gathered from reliable sources such as research datasets and repositories. To avoid bias in model training and achieve accurate results, the dataset is balanced.

C. Permissions Extraction

Permissions are extracted from APK files using static analysis techniques.Each APK contains an AndroidManifest.xml file that holds a list of requested permissions, which is critical in the detection of malware since some permissions can be used to gain access by malicious applications. Permissions are crucial features for detecting malware, as certain permissions (e.g., SEND_SMS, WRITE_EXTERNAL_STORAGE) are often misused by malicious apps.

Permissions Overview:

- 1: Permission is requested.
- 0: Permission is not requested.

The permissions identified are 428, forming the initial feature space for analysis.

Fig. 2. Permissions Extraction Process

D. Feature Selection Using Genetic Algorithm (GA)

Effective feature selection is a critical factor towards achieving higher model accuracy and efficiency, thereby lowering the computational costs. The initial feature space of 428 permissions is reduced, as many permissions are redundant or irrelevant. The study uses a Genetic Algorithm to select the most discriminatory permissions for accurate malware identification.

Steps in Genetic Algorithm:

1) Initialization: A randomly generated population of feature subsets is created. Each subset corresponds to a unique combination of permissions.
2) Measuring Subset Quality: Fitness of every subset measured in terms of finding the SVM classifier's accuracy for classification on that particular subset through learning on it.
3) Selection: The best-performing subsets are selected for the next generation.
4) Crossover: Selected subsets are combined to produce new subsets, simulating biological reproduction.
5) Mutation: Random changes are introduced to subsets to explore additional features.
6) Termination: The process continues until the optimal feature subset is found.

E. Model Training

The refined feature set is used to train two machine learning models: SVM and ANN.

Support Vector Machine (SVM):

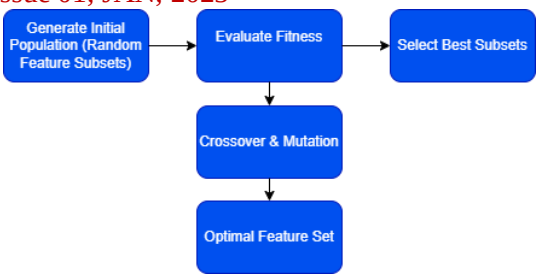

Fig. 3. Genetic Algorithm Process

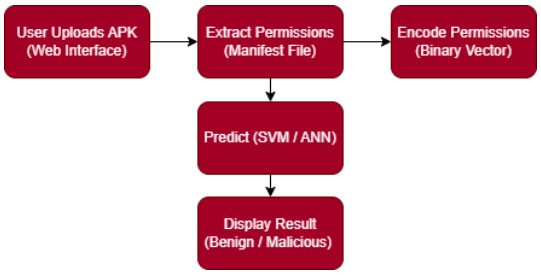

Fig. 5. Real-Time Deployment Workflow

- Purpose: Used as a baseline model for handling high-dimensional datasets.
- Kernel: Radial Basis Function (RBF) for handling non-linear relationships.
- Optimization: Hyperparameters (e.g., C, gamma) are tuned using grid search.

Artificial Neural Network (ANN):

- Purpose: Models the complex interdependencies and non-linear effects which exist in permissions data.
- Architecture:
  - Input Layer: 428 nodes or the reduced subset from GA.
  - Hidden Layers: Four layers with 256, 128, 128, and 32 nodes, using ReLU activation.
  - Output Layer: One node with sigmoid activation for binary classification.
- Regularization: To avoid overfitting, we used dropout with the dropout rate of 0.2.
- Training: Utilizes binary cross-entropy loss with Adam optimizers.

Fig. 4. Model Training Workflow

F. Real-Time Deployment

The trained models are deployed in a Flask-based web application for real-time analysis of uploaded APK files.

Workflow for Real-Time System:

1) APK Upload: Users upload APK files through a web interface.
2) Permissions Extraction: The system extracts permissions from the APK's manifest file.
3) Feature Encoding: Extracted permissions are encoded into a binary vector based on the selected feature subset.
4) Model Prediction: The SVM or ANN model classifies the APK as malicious or benign.
5) Result Display: Outputs the classification result along with a confidence score.

IV. Results and Discussion

A. Performance Metrics

The SVM and ANN models' performances are presented through several key evaluation metrics particularly below:

- **Accuracy:** The general correctness of the predictions.
- **Precision:** The accuracy of positive predictions, calculated as true positives divided by total predicted positives.
- **Recall:** The proportion of correctly identified positives among all actual positive instances.
- **F1-Score:** This metric provides a unified measure of model performance by harmonizing precision and recall.

TABLE I
Comparison of Performance Metrics

Metric	SVM	ANN
Accuracy	91.8%	94.2%
Precision	90.7%	93.4%
Recall	89.6%	92.8%
F1-Score	90.1%	93.1%

B. Confusion Matrices

Confusion matrices break down prediction results into four quadrants: correct identifications (TP and TN) and incorrect identifications (FP and FN).

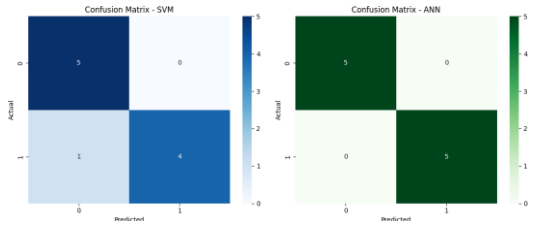

Fig. 6. Confusion Matrices for SVM and ANN

C. Feature Importance

It presents the most critical permissions contributing to malware detection and enhances insights into the decision-making process of the model, hence model explainability.

Top permissions identified include:

- **SEND_SMS:** Associated with SMS fraud.
- **RECEIVE_BOOT_COMPLETED:** Enables persistence after reboot, a common malware tactic.

- **WRITE_EXTERNAL_STORAGE:** Frequently exploited for modifying or encrypting files.

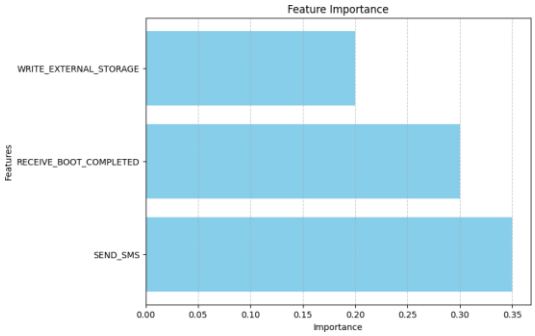

Fig. 7. Feature Importance Analysis

D. Artificial Neural Network (ANN) Architecture

The ANN model is optimized with non-linear patterns in permission data. It consists of:

- **Input Layer:** 428 nodes representing features of permissions.
- **Hidden Layers:** Four layers with ReLU activation:
  - Layer 1: 256 nodes
  - Layer 2: 128 nodes
  - Layer 3: 128 nodes
  - Layer 4: 32 nodes
- **Output Layer:** The model utilizes a solitary node with sigmoid activation to facilitate binary classification outcomes.
- **Dropout Regularization:** Prevents overfitting by adding dropout layers after each hidden layer.

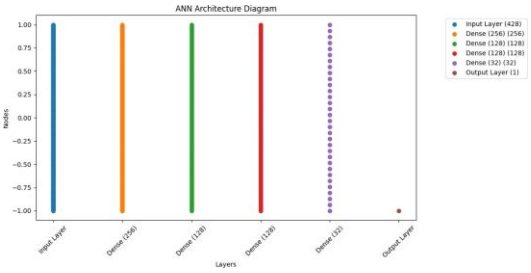

Fig. 8. Artificial Neural Network (ANN) Architecture

E. Observations

- **ANN Outperforms SVM:** ANN achieves higher accuracy and F1-score, demonstrating better capability to model non-linear interactions.
- **Feature Optimization:** GA effectively reduces feature dimensionality while maintaining performance.
- **Real-Time Viability:** Flask deployment ensures scalability and usability for malware detection in practical scenarios.

F. Final Results

The hybrid framework demonstrates superior malware detection performance for Android applications with the following outcomes:

1) **ANN Outperforms SVM:** The ANN demonstrates its capability of effectively capturing non-linear interactions with an accuracy of 94.2% and precision of 91.8%.
2) **Feature Optimization via GA:** GA reduces permissions from 428 to an optimal subset, improving computational efficiency without compromising accuracy.
3) **Practical Real-Time System:** The Flask-based application allows seamless real-time APK analysis, providing actionable insights to users.

V. Conclusion and Future Work

This research proposed a hybrid machine learning framework for detecting and preventing cyber fraud in Android applications. By combining Support Vector Machines (SVM) and Artificial Neural Networks (ANN) with feature selection using Genetic Algorithms (GA), the framework achieved high detection accuracy and efficiency. The ANN model outperformed the SVM baseline with an accuracy of 94.2%, showcasing its capability to model non-linear interactions in the data.

The use of GA successfully reduced feature dimensionality, minimizing computational overhead while retaining predictive performance. The real-time deployment of the framework through a Flask-based web application demonstrated its practical applicability, enabling end-users to detect malware effectively.

Key Contributions:

- **ANN Integration:** Enabled the framework to model non-linear relationships in permission data, outperforming traditional models like SVM.
- **Feature Optimization via GA:** Successfully reduced the dimensionality of the feature space without affecting accuracy and with the reduction of computational overhead.
- **Real-Time Deployment:** Developed a user-friendly Flask-based application ensuring practical relevance and scalability.

Future Work:

- **Integration of Dynamic Analysis:** Incorporating runtime behavioral characteristics, such as API calls and system logs, to enhance detection capabilities.
- **Ensemble Models:** Exploring ensemble techniques to combine multiple machine learning models for improved robustness and accuracy.
- **Expanded Dataset:** Successfully reduced the dimensionality of the feature space without affecting accuracy and with the reduction of computational overhead.
- **Cross-Platform Support:** Adapting the framework for malware detection in other mobile operating systems, including iOS.

This research concludes by introducing a highly effective, scalable, and efficient machine learning-based solution for detecting Android malware. This innovative system significantly

strengthens mobile ecosystem security and sets the stage for pioneering advancements in malware detection and prevention technologies.

REFERENCES

[1] Zhuo Chen, L. Wu, Y. Hu, J. Cheng, Y. Hu, Y. Zhou, Z. Tang, Y. Chen, J. Li, and K. Ren, "Lifting the Grey Curtain: Analyzing the Ecosystem of Android Scam Apps," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 1–15, 2021.

[2] F. Chollet, *Deep Learning with Python*, 1st ed. Manning Publications, 2018.

[3] A. Gupta, A. Malhotra, and K. Sharma, "Machine learning techniques for Android malware detection based on permissions," *Journal of Cybersecurity Applications*, vol. 54, no. 1, pp. 1–15, 2020.

[4] N. Patel and M. Kumar, "Optimizing Android malware detection using Genetic Algorithms for feature selection," *International Journal of Computer Applications*, vol. 178, no. 23, pp. 35–42, 2019.

[5] A. Souri and H. Hosseini, "Comprehensive analysis of machine learning approaches in malware detection," *Human-centric Computing and Information Sciences*, vol. 8, pp. 1–20, 2018.

[6] Canadian Institute for Cybersecurity, "AndMal2019 dataset for Android malware analysis," [Online]. Available: https://www.unb.ca/cic/datasets/invesandmal2019.html.

[7] H. Gascon, D. Arp, and K. Rieck, "Using graph structures for malware detection in Android applications," *Proceedings of the ACM Security Conference*, pp. 1–10, 2013.

[8] G. Suarez-Tangil, J. Tapiador, and P. Peris-Lopez, "Advanced classification techniques for Android malware families based on code structures," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1104–1117, 2014.

[9] J. Sahs and L. Khan, "Static analysis and machine learning for Android malware detection," *Proceedings of the European Security Conference*, pp. 141–147, 2012.

[10] Y. Zhou and X. Jiang, "A comprehensive study on Android malware evolution and characteristics," *IEEE Symposium on Privacy and Security Research*, pp. 95–109, 2012.

[11] S. Kumar and D. S. Gill, "Static analysis techniques for permissions-based Android malware detection," *International Journal of Data Science Applications*, vol. 9, pp. 45–55, 2021.

[12] R. Mehta, "Integration of machine learning in combating Android malware," *Journal of Mobile Security Studies*, vol. 10, pp. 15–25, 2020.

[13] A. Williams and T. Smith, "Comparative analysis of static and dynamic analysis for Android malware detection," *Cybersecurity Advances Proceedings*, pp. 67–78, 2019.

[14] "Androguard documentation," A comprehensive guide on analysis tools for an application developed on Android. [Online]. Available: https://androguard.readthedocs.io/en/latest/.

[15] F. Alajmi and A. Alsulami, "Hybrid Feature Based Approaches in Smart Device Malware Detection for Energy Awareness through Android," *Procedia Computer Science Journal*, vol. 170, pp. 832–838, 2020.