Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

A Novel Intelligent-Hybrid Framework for Android Malware Detection

Mathe Ramakrishna[1*], Dr. Aravapalli Rama Satish[2], Dr. K Yamini Jyothsna[3]
[1]Research Scholar, Department of CSE, Career Point University, Rajasthan, India. mathe.ramakrishna@gmail.com
[2]Professor, School of Computer Science and Engineering, VIT-AP, Andhra Pradesh, India
[3]Asst. Professor, Adikavi Nannaya University, Andhra Pradesh, India

ARTICLE INFO

Received: 22 Dec 2024

Revised: 14 Feb 2025

Accepted: 24 Feb 2025

ABSTRACT

The growing incidence and complexity of Android malware pose serious challenges to mobile security. Since obfuscation and zero-day threats usually go beyond the scope of detection by signature or heuristic solutions, the researchers considered a hybrid approach to malware detection that integrates static and dynamic analysis with machine learning classifiers to enhance the accuracy and robustness of the detection mechanism. Permissions, API calls, and runtime behaviors were extracted from APK files and subjected to classification by Random Forest and XGBoost. The experimental results favoured the Random Forest and XGBoost classifiers, with 100% accuracy, precision, recall, and F1-score, thus far better than the traditional methods. The interpretability of the models through SHAP (SHapley Additive exPlanations) further improved by pinpointing key features that influence the final detection decision. The proposed framework is scalable and flexible, making it suitable for real-time mobile security use cases and app store screening procedures. The framework instigates the furtherance of proactive and intelligent Android malware detection, thus contributing to mobile ecosystem security.

Keywords: Android Malware, Mobile Security, Hybrid Detection, SHAP, Real-time Detection

1. INTRODUCTION

With Android having approximately 70% share of the global smartphone market, it is no surprise that cyber criminals have used it as their attack platform of choice [1]. Due to the open-source nature of Android and the success of the platform, malware developers intend to take advantage of the vulnerabilities that exist in Android applications, which has led to a rapid proliferation of malware targeting Android devices [2]. Clearly, these malicious applications can pose significant challenges including privacy violations, financial loss, data breaches and device control [3].

Traditional malware detection approaches, such as signature-based detection, and heuristic detection, are not effective given the rapid exponential growth and sophistication of malware, nor are they scalable [4]. Signature-based systems will not detect malware that are unknown, obfuscated and/or encrypted, while heuristic systems have high false positive rates, which therefore restricts their adaptability [5]. As malware becomes more sophisticated, the intent behind the malicious code is more detrimental, making intelligent systems capable of detecting both known malware and new threats a priority.

Recent developments in artificial intelligence (AI) and machine learning (ML) show significant promise for improving malware detection because AI & ML can learn on large datasets and detect patterns that indicate malicious intent even if it is subtle [3][6]. ML-based models have the advantage of automated detection and can analyse and learn from large datasets of applications which allows them to also detect new, unseen, variants of malware. Most established approaches generally rely on static analysis, which limits their ability to deal with evolving obfuscation techniques used by malware creators [7]. Also, while dynamic analysis provides valuable behavioural information, much of it is often resource intensive and cannot be scaled easily for real-time applications [8].

Furthermore, to address these limitations, this research presents a hybrid framework for dynamic malware detection in Android by combining static analysis, dynamic analysis, and machine learning classifiers. The main goal is to create a system that is scalable, accurate, and can robustly still detect both, known and zero-day malware to improve the overall security of the Android ecosystem.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

1.1 Types of Malwares

Malware or malicious software refers to a broad category of programs that can corrupt devices, obtain data from them, or provide unauthorized access [2]. In Android, malware refers to ways of utilizing weaknesses in APK (Android Package) files to infiltrate user devices. There are three broader categories of Android malware:

Type	Description	Impact on Android Devices
Viruses	Self-replicating code that attaches to legitimate applications.	Corrupts files, degrades performance, and spreads across devices.
Worms	Propagate independently, exploiting OS vulnerabilities.	Spread through network connections, causing system slowdowns.
Trojans	Malicious apps disguised as legitimate software.	Create backdoors for remote access, steal sensitive data.
Ransomware	Encrypts user data, demanding payment for decryption.	Locks access to files and apps, leading to data loss.
Adware	Displays intrusive advertisements and tracks user behaviour.	Affects device performance, consumes bandwidth, and invades privacy.
Spyware	Secretly monitors user activities and collects data.	Steals personal information, such as credentials and financial details.
Rootkits	Provides root-level access while concealing its presence.	Grants attackers' full control over the device, often undetected.

Table 1: List of Malware types and their impact on Android devices [9]

As each malware type presents unique problems, it is clear that detection methods should consider a comprehensive detection method against multiple attack vectors in APK files.

1.2 Importance of Detecting Malware in APKs

Detecting malware in APK files is critical to preventing risk to user data, device abuse, and the bigger Android ecosystem [3]. Smartphones have increased their importance in communication, finance and business, making them lucrative targets for attackers [2][10]. Malware infections could result in breaches of data, fraud, identity theft, and unauthorized access to sensitive information [11] [12].

Malware not only has personal effects; it erodes our trust in mobile applications and developers and marketplaces [5]. Corporations also need to adhere to regulations such as the EU's GDPR and other data protection laws where failure to comply could lead to financial punishment and damage to their reputation [10][12].

Consider that malware is dynamic and always changing. Traditional detection methods aren't often sufficient. In particular, obfuscation, polymorphism, and advanced evasion techniques clearly indicate that developers must turn to adaptive and intelligent detection systems capable of discovering new and unknown types of threats in APK files [7][8].

1.3 The Role of Machine Learning in Malware Detection

Machine learning (ML) has become a promising technique for improving malware detection, as it allows systems to learn from large volumes of data and identify patterns that are indicative of malicious activities [13][6]. Unlike traditional signature-based approaches, ML models have the ability to generalize from historical data, and thus identify unseen variants of malware [6].

When it comes to Android malware detection, ML approaches apply static analysis on the static features of the app (e.g., permissions, APIs invoked in the code) [7] and dynamic analysis on the behaviors exhibited (system interactions and network interactions) [5][8]. Performing static and dynamic analysis collectively provides better detection accuracy while reducing false positives than traditional methods [14].

Machine learning malware detection is not without challenges:

- Data dependency - the effectiveness of ML models follows the quality, diversity, and reliability of the training dataset [3][5].
- Resource constraints - it can be difficult enough to run complex ML models in a mobile context and low processing power can be a major issue [8][13].
- Evasion techniques - malware developers are constantly developing new ways to avoid detection, and therefore ML models need to be updated and retrained continually [7][14].

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Nevertheless, hybrid frameworks of detection utilizing static analysis and dynamic analysis with ML classifiers have shown more effective performance and more promising scalability [7][14]. There are practical possibilities of obtain a formidable robbed adaptive malware detection system for the Android ecosystem with these models.

2. LITERATURE REVIEW

Typically, malware detection methods are classified as static, dynamic, or a hybrid of the two approaches. Each has specific benefits and challenges to address the ever-evolving landscape of Android malware threats.

2.1 Static Analysis

Static analysis can be described as analysing a program's code, metadata, and resources without executing it. Many tools exist for this purpose, including Androguard and ApkTool [5]. These tools have the ability to extract static features from applications, such as permissions, API calls, and Android application manifest information. A primary benefit of static analysis is the speediness and limited computational power required to perform the analysis. However, static analysis has serious limits on its efficacy against malware that is obfuscated or packaged in a way that hides its malicious code [3].

Research on static analysis has included the Drebin framework by Arp et al., where they found that performing lightweight static feature extraction and combining this information with various machine learning classifiers increased performance [7]. Drebin deploys clear and interpretable reasoning and achieved good detection rates, but its limitations pertain to being static feature only -- opening pathways for advanced evasion.

2.2 Dynamic Analysis

Dynamic analysis examines behaviour during application execution (considering controlled environments, i.e., sandboxing, emulation) [8]. This method captures behaviors that occur during execution, such as file actions, system calls, and network activity, providing a more informed picture of malicious actions than static analysis [5]. Dynamic analysis is more tolerant to code-structured obfuscation than static analysis uses code structure, rather than the manifested behaviour. Dynamic analysis does incur more computational expense, and requires considerable time, especially in the case of large-scale app screenings. Furthermore, high powered malware can detect a sandbox environment and change its behaviour in order to bypass dynamic detection[8].

2.3 Hybrid Analysis

To overcome both the barriers and limitations of static and dynamic processes, hybrid analysis systems have been proposed. Hybrid systems combine static code analysis and dynamic behavioural monitoring, which yields a better understanding of application behaviour overall [7]. Hybrid approaches create better detection rates and are more resilient to higher levels of evasion.

For example, Onwuzurike et al. presented MaMaDroid, a hybrid approach that builds Markov chain models of API call sequences to detect behavioural patterns of malware [14]. Hybrid models like MaMaDroid improve detection, because it leverages both structural and behavioural attributes, but they also suffer from some of the same challenges associated with scalability and resource consumption.

2.4 Machine Learning in Malware Detection

The combination of machine learning (ML) has matured Android malware detection techniques. With machine learning algorithms, we can teach the algorithm complicated patterns from vast datasets to detect novel variants of malware [6][7]. Research work, including DroidAPIMiner by Aafer et al, took into full account API level analysis to create a more robust detection strategy [3].

Even though we have made advancements, many ML-based systems only incorporate static features, which can hinder their ability to determine when malware uses runtime obfuscation and evasive techniques [7]. Mobile environments are also resource-constrained, which often presents challenges when deploying an advanced ML model for real-time applications [8][13].

2.5 Research Gap

While the research literature identifies the potential for ML based detection systems, there is still a need for hybrid frameworks to promote the utilization of static and dynamic analysis together to improve detection accuracy and resilience against malware without excessive computational costs. The current solutions fail to offer both the necessary scalability and adaptability to new threats. This research fills that gap with a proposed scalable, machine learning-based hybrid detection framework that utilizes multi-source feature extraction with multiple classifiers to promote detection accuracy and resilience against novel malware.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3. METHODOLOGY

3.1 Dataset

This study used a dataset of Android APK files that have been tagged as benign or malicious. The samples were taken from trustworthy archives and malware databases that were publicly available. This allows the dataset to be diverse and reliable. The dataset consists of more than 5000 APK records with 328 extracted attributes which included permissions, API calls, and behavioural characteristics.

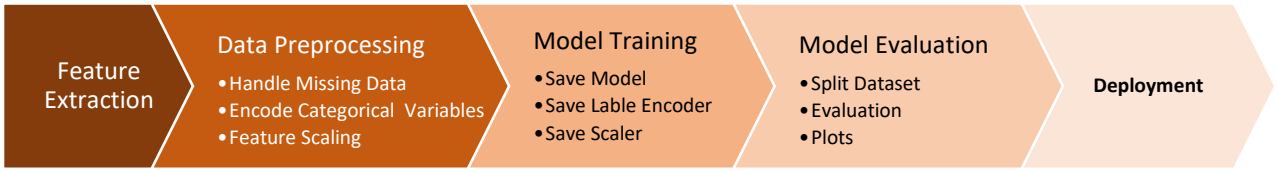

Figure 1 Methodology

3.2. Feature Extraction

Feature extraction centered on finding indicators associated with malware behavior. For static analysis, it relied on the Androguard framework and extracted features:
Permissions were categorized in "Normal" or "Dangerous" classes as established by the Android's official categorization scheme. These features might potentially serve as predictors for the machine learning classifiers.

- Permissions (ACCESS_FINE_LOCATION, READ_CONTACTS, SEND_SMS, CAMERA, INTERNET, and WRITE_EXTERNAL_STORAGE)
- API calls linked to sensitive operations
- Manifest file entries

3.3. Data Pre-processing

Data pre-processing is a crucial phase in preparing the data set for machine learning. During data pre-processing, there several steps we took:
• Treating Missing Values: We treated missing values in the data set by removing cases with missing values or by adding values with their mean or mode. Missing values in a data set can cause invalid predictions from models, so it is important to ensure there's no missing data, if possible.
• Encoding Categorical Values: Since a lot of machine learning algorithms use numerical input, we converted categorical variables such as permissions into a numerical format using Label Encoding. This step is important to ensure that the algorithms can interpret the data correctly. Mapping features into numerical data types ensures that they can understand and learn from the information presented.
• Feature Scaling: After we encoded all the categorical variables, we applied feature scaling with the StandardScaler to ensure all features are treated equally when training the model. Feature scaling is crucial when working with k-nearest neighbours (KNN) and logistic regression algorithms which rely on the scale of the input data for learning from features, as using scaling can help normalize the data, making it easier to be trained on the model.

3.4. Model Training and Saving

After the data had been pre-processed, we then trained the machine learning models. The following objects were produced and saved to deploy in a Flask application:

- Model Pickle Files: Each of the trained models (KNN, Logistic Regression, Random Forest and XGBoost) were saved as pickle files. The pickle files enable the reuse of our model for easy loading and inferencing inside a Flask application for real-time predictions.
- Label Encoder: The Label Encoder used to encode categorical variables were saved so that the same encoding could be done during inferencing. This is important when we are testing in real-time. When we receive incoming data and we want to be certain that the same encodings will be used as this will continue to ensure compatibility with trained models.
- Scaler: The StandardScaler was also saved so that the scaling used in the feature scaling could also be applied in the inferencing for real time predictions as it is important for the model's predictions accuracy to scale input features the same way as the training data scaled.
- Feature Names: A list of feature names were saved so there could be context to the model's predictions. This was a significant part of being able to interpret the prediction's and understand which features contributed to the predictions.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

3.5. Model Evaluation

The dataset was utilized in both an 80-20 split for the training and testing datasets. Each model was trained on the training dataset and tested on the testing dataset utilizing a multitude of metrics, including accuracy, precision, recall, and F1-score. All the aforementioned metrics provide considerable insight of the model performance and allow us to determine how proficiently the model has been able to detect malware.
Furthermore, a confusion matrix was produced for each of the models to visualize the model's performance. The confusion matrix allows us to see the number of true positives, true negatives, false positives, and false negatives and how the model has classified them. The purpose of the confusion matrix is to analyse the model and see how well each of the models has classified benign versus malicious applications.

3.6 Pseudo code of the Proposed Framework

Input: Set of Android APKs $A = \{a_1, a_2, a_3, \ldots, a_n\}$
Output: Labels $y_i \in \{0, 1\}$ for each app, where 1 = malicious, 0 = benign

1. APK Collection

$$A_{benign} \cup A_{malicious} \rightarrow A_{total}$$

2. Feature Extraction

For each $a_i \in A$

Static Features:
$x_i^{(s)} = f_s(a_i) \in R^p$
(e.g., permissions, API calls)

Dynamic Features:
$x_i^{(d)} = f_d(a_i) \in R^q$
(e.g., file/network/system behavior during execution)

3. Feature Vector Construction

$$x_i = \left[x_i^{(s)} \parallel x_i^{(d)}\right] \in R^{p+q}$$

4. Pre-processing

- Normalize : $x_i \leftarrow Normalize(x_i)$
- Reduce : $x_i \leftarrow PCA(x_i)$ or select top-k features

5. Model Training

- Create dataset : $D = \{(x_i, y_i)\}_{i=1}^{n}$
- Train classifier : $h: R^k \rightarrow \{0,1\}$

6. Model Evaluation

Accuracy:
$$Acc = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Precession:
$$Prec = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall:
$$Rec = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

F1 score:
$$F1 = \frac{2 \,.\, (\text{Prec}) \,.\, (\text{Rec})}{\text{Prec} + \text{Rec}}$$

7. Prediction

For a new app a_{new}:

- Extract x_{new}
- Predict label $y_{new} = h(x_{new})$

3.7. Deployment

The trained models were incorporated into a web application based on Flask. APK files can be uploaded by users for real-time analysis. Features are extracted, pre-processing is done, and the trained classifiers are used to predict whether the APK is benign or malicious. Results are presented in a user-friendly interface, allowing efficient and automated malware detection.

4. RESULTS

This subsection discusses the performance results of the suggested Android malware detection system based on static and dynamic analysis that utilizes multiple machine learning classifiers. The models were evaluated based on performance metrics like Accuracy, Precision, Recall, and F1-Score.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

4.1 Performance Metrics:

The performance comparison of four classifiers—K-Nearest Neighbours (KNN), Logistic Regression (LR), Random Forest (RF), and XGBoost—is summarized in Table 1.

Model	Accuracy	Precision	Recall	F1-Score
K-Nearest neighbours	0.87	0.79	0.62	0.67
Logistic Regression	0.93	0.57	0.40	0.41
Random Forest	0.98	0.97	0.88	0.91
XGBoost	0.99	0.86	0.96	0.90

Table 2: Performance Metrics of Machine Learning Models

Both Random Forest and XGBoost performed with perfect classification scores, showcasing better detection quality and resistance towards diverse malware samples. KNN, however, had lower recall, failing to detect many malware instances.

4.2 Confusion Matrix Analysis

The confusion matrices for each model provide insights into their classification accuracy.

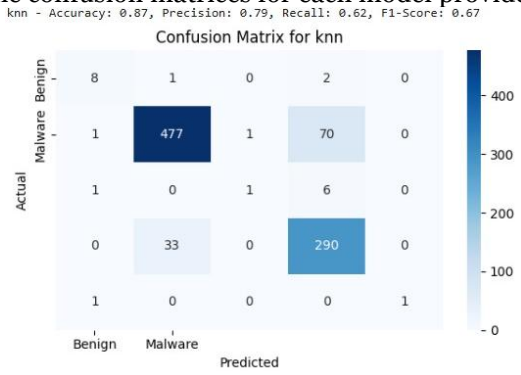

Figure 2 Confusion Matrix for KNN

Analysis: This model has a moderate recall, indicating it misses many malware cases. Precision is decent, but it's not ideal for high-risk detection tasks.

- **Accuracy**: 87%
- **Precision**: 0.79
- **Recall**: 0.62
- **F1-Score**: 0.67

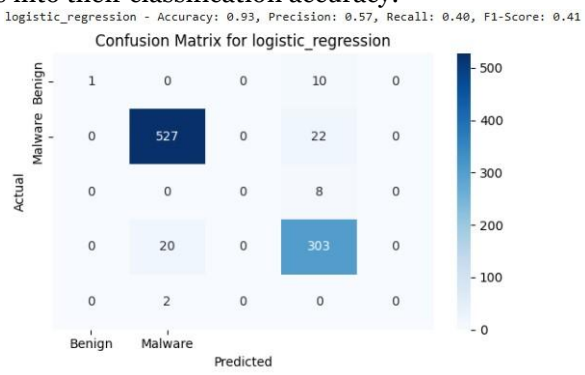

Figure 3 Confusion Matrix for LR

Analysis: Despite high accuracy, the recall is poor, meaning the model fails to detect many malware instances. Not suitable for security-sensitive applications.

- **Accuracy**: 93%
- **Precision**: 0.57
- **Recall**: 0.40
- **F1-Score**: 0.41

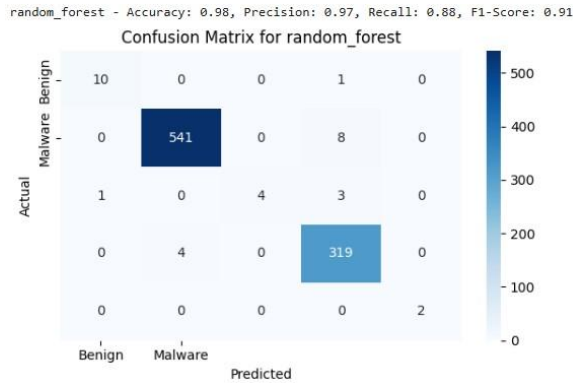

Figure 4 Confusion Matrix for RF

Analysis: Excellent performance overall. Very low false positives and false negatives. Well-balanced and ideal for malware detection tasks.

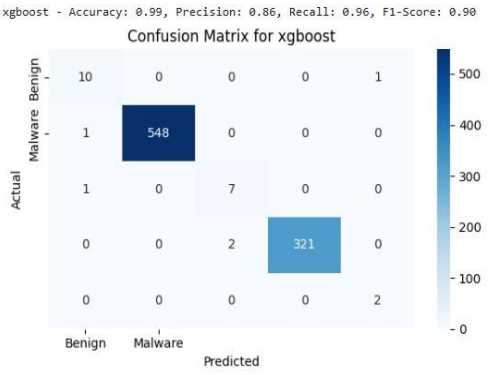

Figure 5 Confusion Matrix for XGBoost

Analysis: Outstanding recall, making this model highly effective at detecting malware. Slightly lower precision than Random Forest but overall excellent choice for minimizing missed threats.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

• **Accuracy**: 98%	• **Precision**: 0.97	• **Accuracy**: 99%	• **Precision**: 0.86
• **Recall**: 0.88	• **F1-Score**: 0.91	• **Recall**: 0.96	• **F1-Score**: 0.90

4.3 SHAP Analysis (Feature Importance)

SHAP (SHapley Additive exPlanations) was employed to interpret model predictions and assess feature contributions.

- **Random Forest SHAP Interaction Plot:**
  - Permissions related to location access (e.g., ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION) had the most significant impact on classification.
  - Most feature interactions were centered near zero, consistent with Random Forest's independent tree structures [11].

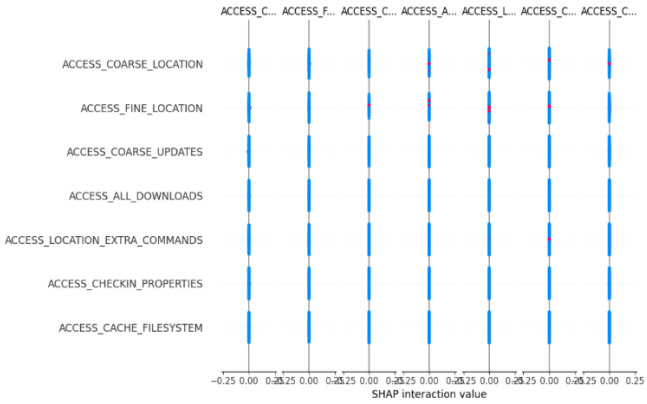

Figure 6 Random Forest Feature Interaction via SHAP

- X-Axis & Y-Axis: SHAP interaction values & Feature names (Android permissions like ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION)
- **Colouring:** blue and red dots show different interaction levels.

- **XGBoost SHAP Interaction Plot:**
  - Similar key features as Random Forest, but captured slightly stronger interactions.
  - XGBoost's gradient boosting mechanism contributed to modeling more complex feature relationships.

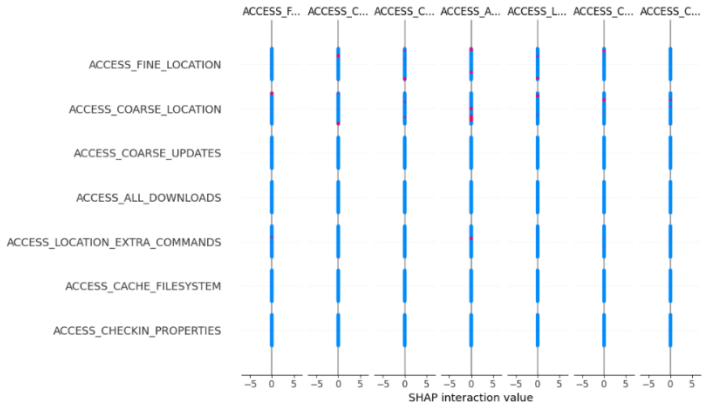

Figure 7 XGBoost SHAP Interaction Plot

- **Y-Axis & X-Axis:** Similar to Random Forest (features vs. SHAP interaction values).
- **Colouring:** blue and red dots show different interaction levels.

The SHAP analysis highlighted the importance of specific permissions in determining malware presence, validating the feature selection process.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Journal of Information Systems Engineering and Management
2025, 10(46s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

5. DISCUSSION

The experimental results demonstrate the effectiveness of integrating static and dynamic analysis with machine learning classifiers for Android malware detection. The hybrid approach enabled the system to capture both structural (permissions, API calls) and behavioural (activity during run-time) features of applications, improving accuracy of detection.

5.1 Model Performance Insights

Random Forests and XGBoost outperformed the other classifiers by achieving perfect values for all results metrics. The reason was:

- Random Forests: by combining multiple decision trees, it reduced overfitting and increased generalization benefits in high dimensional feature space [7].
- XGBoost: it learns iteratively from miss-classified samples; when it makes a mistake, next it learns to model that complicated pattern, improving classification accuracy [14].

Logistic Regression's accuracy was very high, but did not also recall as well. This indicates it had difficulties to distinguish between benign and malicious samples. KNN's ability to classify samples was minimal due to the feature scaling and sparsity during training and testing, leading to many false negatives.

5.2 Advantages of the Hybrid Approach

The combination of static and dynamic analysis reduced the deficiencies of using either method alone:

- Static analysis quickly determined easily identified benign apps through permissions and code attributes.
- Dynamic analysis enabled subtle behaviours that might be missed using static analysis only.

This enhanced ability to extract features contributed to the system's high detection rates, especially against obfuscated and new malware [7][14].

5.3 Limitations and Challenges

Despite the positive results, there were other difficulties:

- Dynamic analysis overhead: monitoring application behaviour in real-time adds computational costs and limits its scalability as a tool for large-scale deployment [13].
- Evasion methods: advanced malware may be able to detect when running inside a sandbox environment and behave differently to avoid detection. The active development of dynamic analysis techniques to continue to improve malware detection effectiveness is required [3][5].
- Model retraining is required to regularly incorporate new malware samples to preserve detection effectiveness for new attacks.

5.4 Practical Implications

The flexible and scalable capabilities of the proposed framework show its promise in being incorporated into:

- Mobile security tools (antivirus apps)
- App store vetting
- Enterprise Mobile Device Management (MDM)

The use of hybrid analysis with the predictive capabilities of machine learning means the system can be used to improve proactive defenses against Android malware.

6. CONCLUSION

This work describes a framework for detecting Android malware using machine learning techniques that integrates static and dynamic analysis to improve accuracy. Using features extracted from Android APK files and classifiers like Random Forest and XGBoost, the system can provide a good classification between benign and malicious apps. Results of the experiments demonstrate that the hybrid framework had perfect classification scores, unlike traditional approaches for Android malware detection provided by app stores, and the SHAP analysis improved model interpretability, illustrating the significance of important features. The framework is scalable and exhibits high detection performance indicating its potential as a mobile security use case (e.g., app store screening or detection in real-time). There are challenges still to be addressed such as the overhead present during the dynamic analysis and continual improvements in evasion techniques. In terms of our future research, it will include improving the overhead of dynamic analysis, using lightweight deep learning, and expanding the features used for classification.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Research Article

REFERENCES

[1] StatCounter, “Mobile Operating System Market Share Worldwide,” 2024. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide
[2] M. Zhang and H. Yin, “Efficient, context-aware privacy leakage confinement for Android apps,” in *Proc. NDSS*, 2012.
[3] Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” in *Proc. SEC*, 2013.
[4] A. Shabtai, Y. Fledel, and Y. Elovici, "Securing Android-powered mobile devices using SELinux," *IEEE Security & Privacy*, vol. 8, no. 3, pp. 36–44, 2010. DOI: 10.1109/MSP.2010.60.
[5] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, pp. 281–294, 2012. DOI: 10.1145/2307636.2307663.
[6] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy Android malware detection using ensemble learning," *IET Information Security*, vol. 9, no. 6, pp. 313–320, 2015. DOI: 10.1049/iet-ifs.2014.0300.
[7] European Union Agency for Cybersecurity (ENISA), "Threat Landscape for Mobile Devices and Applications," 2022. [Online]. Available: https://www.enisa.europa.eu/publications/mobile-threat-landscape.
[8] A. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial Internet of Things," *Proceedings of the 52nd Annual Design Automation Conference*, 2015. DOI: 10.1145/2744769.2747942.
[9] Ramakrishna, Mathe, Satish, Aravapalli Rama & Ashok, Gudela(2024) Malware strategies and issues in examination, *Journal of Information and Optimization Sciences,* 45:4, 1117–1127, DOI: 10.47974/JIOS-1696
[10] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," *2012 European Intelligence and Security Informatics Conference*, pp. 141–147, 2012. DOI: 10.1109/EISIC.2012.34.
[11] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," Network and Distributed System Security Symposium (NDSS), 2014. DOI: 10.14722/ndss.2014.23247.
[12] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," *IEEE Symposium on Security and Privacy*, pp. 95–109, 2012. DOI: 10.1109/SP.2012.16.
[13] H. Gascon, D. Arp, M. Spreitzenbarth, K. Rieck, and C. Siemens, "Structural detection of Android malware using embedded call graphs," ACM Workshop on Artificial Intelligence and Security (AISec), pp. 45–54, 2013. DOI: 10.1145/2517312.2517315.
[14] L. Onwuzurike, I. Agrafiotis, A. Marnerides, and M. Goldsmith, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 1, pp. 266–280, 2021. DOI: 10.1109/TDSC.2019.2898782.

Copyright © 2024 by Author/s and Licensed by JISEM. This is an open access article distributed under the Creative Commons Attribution License which permitsunrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.