Journal of Cyber Security and Risk Auditing

https://www.jcsra.thestap.com/

A Comprehensive Review of Machine Learning Approaches for Android Malware Detection

Aneesha Davarasan[1], Joshua Samual[1], Kulothunkan Palansundram1[1], Aitiizaz Ali[1]

[1] *Department School of Technology Networks, Security, Forensic Asia Pacific University of Technology & Innovation (APU), Malaysia*

ARTICLE INFO

Article History

Received: 01-11-2024
Revised: 30-11-2024
Accepted: 05-12-2024
Published: 06-12-2024

Academic Editor:

Prof. Youakim Badr

Vol.2024, No.1

DOI:
https://doi.org/10.63180/jcsra.thestap.2024.1.5

ABSTRACT

In today's digital age, smartphones have evolved beyond communication devices, becoming integral to various aspects of daily life. Android, as a leading mobile operating system, dominates the market due to its open-source nature and vast user base. However, this widespread adoption has made it a prime target for increasingly sophisticated malware attacks. Traditional malware detection methods, primarily reliant on signature recognition, have proven insufficient in countering these dynamic threats. This paper provides a detailed review of Android malware detection approaches leveraging machine learning techniques. By examining the underlying Android architecture and security models, we explore static, dynamic, and hybrid analysis methods, highlighting the crucial role of feature selection in improving detection accuracy. Additionally, we address the significant challenges posed by deterioration in detection model performance over time and evasion tactics employed by malware, proposing advanced strategies such as adversarial training and regular model updates to enhance system resilience. This review aims to synthesize current methodologies, offering a critical evaluation and identifying potential avenues for future research to fortify Android malware detection systems.

Keywords: Malware Attacks; Android Malware Detection; Machine Learning; Dynamic Analysis.

How to cite the article
Davarasan, A., Samual, J., Palansundram, K., & Ali, A. (2024). A Comprehensive Review of Machine Learning Approaches for Android Malware Detection. Journal of Cyber Security and Risk Auditing, 2024(1), 39–60. https://doi.org/10.63180/jcsra.thestap.2024.1.5

**Corresponding author. Email:* aitizaz.ali@apu.edu.my

ISSN: 3079-5354

1. Introduction

In today's digital ecosystem, smartphones have transcended their traditional roles as mere communication tools to become indispensable instruments for various aspects of daily life. They now serve critical functions across social interaction, entertainment, productivity, and much more. Among the leading mobile operating systems, Android stands out due to its open-source framework and significant market penetration. As of June 2024, Android holds approximately 71.80% of the global mobile operating system market share, according to StatCounter [1]. This widespread adoption has also made Android a prime target for malicious actors, who continuously exploit its vulnerabilities to propagate malware.

The Android ecosystem is characterized by its open architecture, which, while fostering innovation and development, introduces several points of vulnerability. With over 3 million applications available on the Google Play Store [2], Android's openness poses risks, including the potential for malware infiltration through lenient permission controls and the ability to execute external code. A telling example of this is the identification of 3.25 million new malicious Android applications in 2016 alone by G DATA CyberDefense [3]. The staggering discovery of a new malicious app nearly every 10 seconds underscores the growing scale of the threat, highlighting the limitations of traditional signature-based malware detection methods.

Traditional methods of malware detection, which primarily rely on identifying distinct patterns or signatures of known viruses, are becoming increasingly ineffective in the face of rapidly evolving cyber threats. Modern malware can easily evade these methods by altering portions of their code, effectively rendering them invisible to signature-based detectors. G DATA's report indicates the discovery of a new sample of malicious software every 8 seconds, amounting to about 11,000 new instances daily [3]. This alarming rate of malware proliferation calls for more adaptive and robust detection methods.

Machine learning (ML) offers a promising solution in the fight against malware. By analyzing patterns in existing data, ML models can identify new and unknown threats, making them a powerful tool in the cybersecurity arsenal. Unlike traditional methods, machine learning techniques, implemented through both static and dynamic analyses, provide a more resilient and flexible approach to malware detection. Static analysis involves examining an application's code without executing it, which offers efficiency but might be vulnerable to obfuscation techniques. On the other hand, dynamic analysis observes the behavior of the application during execution, revealing malicious actions that static analysis might miss, although this approach requires more processing resources.

The importance of robust malware detection systems is amplified by the sensitive nature of data stored on smartphones, including personal and financial information. As smartphones become further integrated into essential activities ranging from financial transactions to personal health monitoring the stakes of malware attacks rise correspondingly. Malware targeting banking applications can lead to significant financial losses, while spyware can result in severe privacy breaches. This makes the development of more sophisticated detection techniques not only necessary but crucial.

Despite numerous studies exploring the use of machine learning in detecting Android malware, many focus on specific methodologies with limited scope. For instance, some surveys concentrate solely on deep learning techniques [4], while others provide a restricted review of research from particular years [5]. A more comprehensive analysis spanning a broader range of machine learning methodologies and examining their practical implications is necessary to drive progress in this field.

As malware continues to grow in complexity, traditional detection methods struggle to keep pace. Polymorphic malware, which can alter its code to avoid detection, presents a significant challenge to static signature-based techniques. Similarly, zero-day exploits can entirely bypass signature-based detection by exploiting vulnerabilities unknown to software vendors. Machine learning, with its ability to extrapolate from existing data and identify patterns indicative of malicious activity, offers a viable solution to these challenges.

Machine learning detection methodologies can be broadly categorized into supervised, unsupervised, and semi-supervised approaches. Supervised learning relies on labeled datasets to train models capable of classifying applications as either benign or malicious, necessitating substantial amounts of labeled data for high detection accuracy. In contrast, unsupervised learning does not require labeled data and can detect anomalies by identifying deviations from normal behavior. Semi-supervised learning combines both approaches, utilizing a smaller labeled dataset alongside a larger set of unlabeled data, thereby balancing detection accuracy with the need for labeled data.

Various machine learning techniques have been employed in Android malware detection, including decision trees, support vector machines, neural networks, and ensemble methods. Each method comes with its own set of strengths and limitations. For example, decision trees are interpretable and efficient but may be prone to overfitting, while neural networks, particularly deep learning models, excel at identifying complex patterns but require large datasets and computational resources for effective training.

Feature extraction and selection are critical steps in applying machine learning to malware detection. Features can be extracted from static characteristics of the application, such as permissions, API calls, and code structure, or from dynamic behaviors observed during execution, such as system calls, network activity, and resource usage. Effective feature selection is crucial for enhancing the performance and efficiency of machine learning models by focusing on the most relevant features from a potentially large set.

Evaluating the effectiveness of machine learning-based malware detection systems involves considering various metrics such as accuracy, precision, recall, F1-score, and the area under the receiver operating characteristic curve (AUC-ROC). These metrics provide insights into the model's performance, particularly its ability to correctly identify malware (true positives) while minimizing false alarms (false positives). Additionally, the system's robustness against evasion techniques such as code obfuscation and adversarial attacks is a critical consideration.

However, several challenges persist in implementing machine learning-based malware detection systems. One significant challenge is the evolving nature of malware, which continuously adapts to evade detection methods. This necessitates ongoing research and development to modify and improve detection algorithms. Another challenge is the threat of adversarial attacks, wherein malware creators deliberately alter their code to deceive machine learning models. Addressing these challenges requires a multifaceted approach, including developing more robust models, incorporating adversarial training techniques, and integrating threat intelligence.

This review aims to provide a comprehensive analysis of machine learning-based techniques for detecting Android malware. We systematically examine the methodologies, datasets, and outcomes of various studies, categorizing them into static, dynamic, and hybrid analysis approaches. Our review covers a broad spectrum of machine learning techniques, including feature selection, data preprocessing, evaluation of detection effectiveness, and resilience against evasion techniques.

Furthermore, this paper addresses the limitations and challenges inherent in current machine learning-based malware detection systems, including high computational requirements, the impact of evasion attacks, and the need for regular updates to maintain effectiveness. By offering a thorough overview of these issues, this paper seeks to enhance the understanding of machine learning's role in Android malware detection and provide valuable insights for researchers and practitioners in the field of cybersecurity.

2. Method of Literature Collection

In undertaking a comprehensive literature review on Android malware detection approaches based on machine learning, it is imperative to meticulously outline the paper selection criteria to ensure the representativeness and credibility of the information sourced. The procedure used for selecting papers in this review is detailed as follows:

2.1 Defining the Scope of Collection

The initial step in our methodology involved defining the precise scope of the literature collection. Given the dual focus on Android malware detection and machine learning techniques, the scope was segmented into several subtopics: static and dynamic analysis methods, feature selection strategies, and the impact of adversarial attacks on detection efficacy.

- **Static Analysis**: Involves examining the code of an application without executing it, identifying features like permissions, API calls, and bytecode sequences.

- **Dynamic Analysis**: Entails executing the application in a controlled environment to observe its behavior, capturing features such as system calls, network traffic, and runtime modifications.
- **Feature Selection**: Encompasses techniques to select the most informative features from the raw data to improve the performance of machine learning models.
- **Adversarial Attacks**: Investigates how malware can evolve to bypass detection mechanisms and the robustness of machine learning models against such attacks.

To enhance the completeness of our search, an incomplete reverse snowball search was performed on the reference lists of articles identified through the initial keyword search. This method ensures that seminal and highly cited papers, which might not be captured through keyword searches alone, are included.

2.2 Keyword Determination

Keywords were meticulously selected to ensure comprehensive coverage of relevant literature. A multi-tiered keyword strategy was employed:

1. **Core Combinations**: Keywords related to the review topic, such as "Android malware detection" + "machine learning" and "Android malware classification" + "machine learning." This approach ensures the inclusion of papers focusing on the integration of machine learning techniques with malware detection.
2. **Feature-Specific Terms**: Given that malware analysis can be categorized into static and dynamic methods, keywords like "static analysis" + "Android malware detection" and "dynamic analysis" + "Android malware detection" were used. This captures literature focusing on specific analysis techniques.
3. **Review and Survey Keywords**: To identify previous reviews and surveys in the domain, keywords like "review" + "Android security" and "survey" + "Android malware detection" were included. This step ensures that we build upon the existing body of summarized knowledge.

2.3 Data Sources

The literature was sourced from reputable academic repositories to ensure the quality and relevance of the collected papers. The primary databases searched included:

- **IEEE Xplore Digital Library**: Known for its extensive collection of engineering and technology research.
- **ACM Digital Library**: A comprehensive resource for computing and information technology literature.
- **SpringerLink**: Provides access to a wide range of scientific documents across multiple disciplines.
- **ScienceDirect**: A leading full-text scientific database offering journal articles and book chapters.
- **Web of Science**: A robust research platform that provides citation indexing for various academic disciplines.

Additionally, widely used third-party platforms such as Google Scholar, ResearchGate, and Academia were utilized to supplement the search. These platforms often include preprints and conference papers not indexed in traditional databases, ensuring a broader scope of literature.

2.4 Time Frame and Relevance

The focus was placed on literature published within the last decade to capture the most recent advancements and trends in the field. The majority of the collected works were from the last five years (2019-2024) to ensure the review reflects current research directions. However, seminal papers predating this period were also included to provide foundational context.

2.5 Exclusion Criteria

To maintain the relevance and quality of the review, certain papers were excluded based on the following criteria:

- **Irrelevant Content**: Papers with titles related to the review topic but whose content was not aligned with the specific focus of this review were excluded. This step ensures that only papers with direct relevance to the topic are included.
- **Insufficient Detail**: Short conference papers with limited detail (less than four pages) and those lacking comprehensive descriptions of their methodologies or results were not considered. This criterion ensures that the included papers provide sufficient information for in-depth analysis.
- **Unverified Sources**: Papers from sources where intellectual property rights could not be confirmed, or those not published in their final versions, were excluded. This step ensures the credibility and reliability of the sources.
- **Duplicate Entries**: Duplicate papers identified across different data sources were verified and excluded to avoid redundancy. This ensures a streamlined and unique collection of literature.

2.6 Backward Snowballing

To further ensure comprehensive coverage, an incomplete backward snowballing method was applied. This involved reviewing the reference lists of papers identified through the keyword search to find additional relevant literature that may not have been captured initially. This approach is particularly useful for identifying seminal papers and highly cited works that form the foundation of the field.

2.7 Manual Selection and Screening

The manual selection process, while thorough, inherently includes limitations such as potential omissions and the lower efficiency of manual searches. Despite these limitations, the selected papers were rigorously screened to ensure high quality and relevance, providing a robust foundation for this review. Each paper was evaluated for its contribution to the field, methodological soundness, and relevance to the review's focus.

The meticulous application of keyword searches combined with backward snowballing ensures a comprehensive and representative collection of literature. The curated set of papers is sufficient to support an in-depth review of the state of research on Android malware detection using machine learning. By continually updating the search criteria and sources, the relevance and quality of the literature collection are maintained, offering valuable insights and a thorough overview of this dynamic field. This structured approach ensures that the review is built on a solid foundation of high-quality, relevant, and recent research, providing a thorough and insightful analysis of the current state and future directions of Android malware detection using machine learning.

3. Literature Review

Machine learning techniques have been instrumental in driving substantial progress in the domain of Android malware detection. Android-based malware has become more prevalent as mobile devices have become an ever-increasing part of individuals' daily lives; therefore, robust detection and mitigation strategies are required. Machine learning is a highly promising solution owing to its capacity to discern malicious behavior patterns from immense quantities of data. This literature review compiles findings from numerous studies that have investigated advanced machine learning techniques for detecting Android malware. Each of these studies has provided distinct contributions and developments to the field. Further, Table1 shows the summary and critical analysis of the literature review**.**

Author [4] conducted a study in 2022 wherein he implemented a novel approach to malware detection on Android through the integration of permissions features, API calls, and application features. Utilizing an extensive compilation of malicious and benign Android APK samples, the research compiled a unique dataset consisting of static API calls and permission features. By applying the Information Gain algorithm, the feature space was diminished to fifty features, which are more practical and germane. To ascertain whether contextual features could be employed to detect Android malware with an estimated 99.4% accuracy, a series of experiments were undertaken. The considerable precision observed highlights the potential of integrating contextual data to improve the detection capabilities of machine learning models.

On the contrary, a study undertaken by author [5] centered on the categorization of malicious software into discrete classifications, including riskware, banking malware, adware, and SMS malware. A feature vector generation technique was suggested, which utilizes system call frequencies as features to extract dynamic behavior patterns of malware. This technique is founded on Huffman encoding. The model underwent evaluation utilizing machine learning and deep learning methodologies, and the Random Forest classifier achieved a detection accuracy of 98.70%. The findings of this study suggest that malware classification can offer further understanding of the malicious patterns displayed by various forms of malware, which may result in the development of more precise and efficient detection methods.

An additional significant development in this field is the MLDroid framework, which introduced in 2020[6]. MLDroid is a web-based platform that automates the process of identifying malware in Android applications through dynamic analysis. A model is constructed by the framework utilizing a variety of machine learning algorithms and particular features. During training on a dataset consisting of more than 500,000 real-world Android applications, the model achieved an exceptionally high malware detection rate of 98.8%. The notable rate of detection underscores the effectiveness of dynamic analysis and stresses the necessity of training machine learning models with massive datasets.

BFEDroid introduced a machine learning-based detection strategy that combines backward and forward methods, exhaustive subset selection, in a 2022 study [7]. By integrating these techniques, BFEDroid enhances the feature selection procedure in machine learning models. In comparison to current feature selection-based methods for malware detection, the proposed technique demonstrated a significantly improved performance in terms of memory consumption, execution time, and precision (99%). The significance of effective feature selection in enhancing the performance of malware detection systems is demonstrated in this study.

In 2023, researchers introduced a multi-stage, lightweight methodology for the detection of Android malware that makes use of non-intrusive machine learning techniques [8]. Prioritizing user privacy, this framework ensures adherence to application licenses and terms of service. The utilization of non-invasive feature extraction and multi-stage analysis significantly reduces CPU utilization by three orders of magnitude and improves the accuracy of malware detection in comparison to current approaches. Although the framework has merits, it still necessitates additional substantiation in practical situations to completely determine its durability and practical applicability.

DroidDetectMW unveiled an intelligent hybrid model that integrates machine learning techniques with static and dynamic analysis to detect Android malware [9]. The framework of the model consists of three phases: preprocessing, feature selection, and classification. Through the integration of static and dynamic analysis, the model offers a holistic perspective on potential threats. The sophisticated methodologies employed are highlighted by the utilization of neural networks that have been enhanced through the implementation of an improved Harris Hawks Optimization (HHO) algorithm. Nevertheless, the research emphasized the need to resolve the computational intricacy of the model and perform further comprehensive evaluations using a wide range of datasets and real-life situations to ascertain its efficacy.

Author investigated the application of genetic algorithms to feature selection in machine learning models intended for the detection of Android malware [10]. The approach they devised exhibited a higher degree of efficacy than conventional techniques, including information gain, through the augmentation of precision and the reduction of machine learning time. The research centered on static analysis utilizing 1,104 features and demonstrated encouraging advancements in performance. The researchers proposed that dynamic analysis be integrated with static analysis to obtain a more comprehensive understanding of malware behavior. It was suggested that additional research be conducted to evaluate the system in real-world settings and to compare it to other innovative feature selection techniques.

Studies are carried out to investigate the application of linear regression to feature selection in an Android malware detection system in an additional study [11]. By decreasing the dimensionality of feature vectors, the research endeavored to enhance the efficacy of malware detection systems. With the utilization of 27 features, a notable F-measure of 0.961 was attained. The method exhibited a praiseworthy equilibrium between precision in detection and efficiency in computation. The research emphasized the possibility of enhanced efficiency and efficacy, suggesting additional verification via comprehensive testing and comparison with alternative feature selection techniques.

In 2023, AndyWar presented an intelligent Android malware detection system that employs behavior analysis and supervised learning, in another study [12]. Proposed voting algorithm performance yielded a system accuracy rate of 97%, highlighting notable precision in the realm of malware detection. The significance of recognizing unauthorized application behavior and malicious API calls was underscored to tackle the pervasive problem of deceptive applications. It is recommended that additional research be conducted in order to assess its durability and ability to adjust to ever-changing malware threats.

Furthermore, in 2023, an investigation was conducted by researchers into the utilization of diverse machine learning and deep learning algorithms for the purpose of detecting Android malware [13]. An evaluation was conducted on various algorithms, including SVM, KNN, LDA, LSTM, CNN-LSTM, and autoencoder, using the CICAndMal2017 and Drebin datasets. In contrast, the LSTM attained 99.40% accuracy with Drebin while the SVM attained 100% accuracy with CICAndMal2017. The obtained results exhibited great potential, providing evidence of the algorithms' effectiveness in detecting malware. The research suggested that these algorithms be evaluated in real-world scenarios and that a more in-depth analysis of the most influential characteristics be conducted.

In 2022, SCREDENT unveiled [14] a real-time anomaly detection and notification system designed to scale and identify targeted malware on mobile devices. By integrating user group profiling and behavior-triggering probabilistic models, SCREDENT improves the efficacy of dynamic analysis. Real-time processing was accomplished by the system by integrating MongoDB and Apache Spark within a lambda architecture. Nevertheless, it is advisable that future research incorporates comprehensive real-world testing and addresses privacy concerns to achieve even greater optimization.

The studies collectively underscore the potential of machine learning to augment the detection of Android malware. The authors illustrate the efficacy of diverse feature selection methodologies, underscore the significance of contextual data, and highlight the advantages of malware classification. Upon contrasting the various methodologies, it becomes apparent that although studies share high accuracy rates, the approaches diverge regarding techniques for feature selection, methods of analysis, and enhancements to efficiency. For example, research places significant emphasis on the incorporation of contextual attributes, whereas study underscores the advantages of classifying malware based on dynamic behavior patterns [13]. Similar to how Mahindru's framework for dynamic analysis differs from emphasis on feature selection to improve accuracy and efficiency [6].

Upon synthesizing these findings, it becomes evident that the incorporation of sophisticated machine learning algorithms, such as genetic algorithms, hybrid models, or contextual feature incorporation, enhances the ability to detect. Furthermore, real-time processing frameworks and non-invasive methodologies effectively tackle pragmatic considerations pertaining to privacy and computational efficacy. The research conducted by [8] , as well as [14]underscores the significance of scalable real-time processing and non-invasive methods, respectively. These studies present avenues for enhancing malware detection while maintaining privacy and performance standards uncompromised.

However, these studies all have several areas that could be enhanced. Conducting a comparative analysis of the various methods would facilitate comprehension of their respective merits and drawbacks, thereby offering a more precise depiction of the present condition of Android malware detection. For example, BFEDroid demonstrated a notable level of accuracy and efficiency through its implementation of exhaustive, forward, and backward subset selection methods. In contrast, MLDroid achieved an exceptionally high detection rate through the utilization of a substantial training dataset in its dynamic analysis approach. Furthermore, an additional synthesis of the results could yield overarching conclusions regarding the efficacy of machine learning methodologies in this domain. Ongoing model updates and retraining are imperative to maintain malware detection systems that are both resilient and effective in the face of their ever-changing characteristics.

In summary, the studies emphasize the wide range of inventive strategies utilized in the detection of Android malware. This underscores the progress achieved thus far as well as the domains that require additional investigation. A comprehensive examination of these studies indicates that although each methodology possesses its own merits, the integration of their observations may result in more integrated and efficacious resolutions. Ongoing integration of these discoveries is essential for the progression of the discipline, with the goal of developing malware detection systems that are resilient and effective in the face of a constantly changing threat environment.

Table 1. Summary and Critical Analysis of the literature review

Research Paper	Machine Learning Technique	Malware Analysis Technique	Detection Accuracy	False Positives/Negatives

[15]	Ensemble Learning, BGWO	Statistical Feature Engineering	96.95% (binary), 83.49% (category)	Not explicitly detailed
[7]	Backward, Forward, Exhaustive Subset Selection	Feature Selection	99%	Suggested need for real-world testing to validate effectiveness
[8]	Non-Invasive Machine Learning Techniques	Multi-Stage, Non-Invasive Feature Extraction	Not specified	Needs more diverse datasets for validation
[16]	Various ML and DL Techniques	Comprehensive Review of Techniques	Various	Lack of real-world testing discussed
[9]	Neural Networks, Harris Hawks Optimization	Static and Dynamic Analysis	Not specified	Computational complexity and real-world testing needed
[17]	Various ML Techniques	Systematic Review of Techniques	Various	Emphasized the need for further reviews due to evolving threats
[18]	Various classifiers including Random Forest	Static and Dynamic Analysis	Up to 89.36%	More in-depth discussion on False Positives needed
[4]	Information Gain-based Feature Selection	Context-Aware Analysis	99.4% with context	Requires real-world testing and detailed computational complexity analysis
[10]	Genetic Algorithm for Feature Selection	Static Analysis	Not specified	Suggested integration of dynamic analysis for comprehensive evaluation
[6]	Deep Learning, Clustering, Decision Tree, Rough Set Analysis	Dynamic Analysis	98.80%	Further real-world testing needed; discussion on privacy implications suggested

[11]	Linear Regression for Feature Selection	Permission-Based Detection	High F-measure of 0.961	More extensive validation required
[12]	Supervised Learning with a Voting Algorithm	Behavioral Analysis	97%	Long-term effectiveness and resilience against new threats need exploration
[5]	Machine Learning, Deep Learning	Huffman Encoding for Feature Vector Generation	98.70%	Discussion on integration of static features suggested
[13]	SVM, KNN, LDA, LSTM, CNN-LSTM, Autoencoder	Benchmark Dataset Analysis	Up to 100%	Testing in real-world scenarios needed
[14]	Machine Learning, User Behavior Profiling	Real-time Anomaly Detection, User Group Profiling	Not specified	Optimization and extensive real-world testing required

4. Background

The Android operating system, which is built on a customized Linux kernel, is the most extensively utilized mobile operating system worldwide. The open-source nature of this technology allows for considerable customization and integration on a wide range of mobile devices, which promotes the development of a diverse ecosystem of apps and services. Nevertheless, the inherent openness of Android also renders it vulnerable to substantial security risks, especially in the shape of malicious software. This section examines the structure of Android, investigates its security procedures, and addresses the ongoing danger presented by malware.

4.1 Architecture of the Android System

The architecture of Android in Figure 1 is intricately crafted as a layered framework, with the Linux kernel serving as its foundation. This kernel, specifically optimized for Android, coordinates essential system operations including memory management, user input processing, and security maintenance.[19] The kernel is responsible for implementing essential security measures such as sandboxing programs and isolating application processes. These measures are critical for safeguarding the system against unauthorized access and malicious attacks.

The Hardware Abstraction Layer (HAL) is positioned directly above the kernel and serves as a vital intermediary between the hardware components of an Android device and the higher software layers. [20]The purpose of this layer is to simplify the intricacies of the underlying hardware, providing a standardized interface for the Android OS to communicate with. HAL's purpose is to ensure that the operating system can work consistently on different devices with different hardware setups, without needing to modify the fundamental software layers. This abstraction is crucial for Android's capacity to expand across the varied ecosystem of mobile devices.

The Android Runtime (ART), which has superseded the previous Dalvik virtual machine, is positioned above the Hardware Abstraction Layer (HAL) and is accountable for the execution of application code.[21] ART improves the effectiveness of applications by converting bytecode into native machine instructions before installation, a procedure referred to as a head-of-time (AOT) compilation. This is distinct from the just-in-time (JIT) compilation employed in Dalvik, where code is compiled on demand during execution. The approach employed by ART minimizes the computational burden during program execution, resulting in improved application speed and decreased energy usage.

In addition to ART, there are native C/C++ libraries that offer numerous vital functionalities for the Android operating system. These libraries provide extensive support for various system functions, including 2D and 3D graphics rendering, database administration, and network connectivity [22]. They play a crucial role in the performance and capacity of the Android platform, ensuring that higher-level application frameworks may operate efficiently and effectively.

The application framework layer, situated at the highest point of the stack, furnishes the essential APIs required for Android development. The developers interact with the system on this layer by utilizing the supplied APIs to access the device hardware and Android system services. The package encompasses all essential components for creating sophisticated and captivating apps, including user interface controls, resource management, and location services. The layer of system apps, including the contacts app, the phone dialer, and system management tools, contribute essential functions to all Android devices.

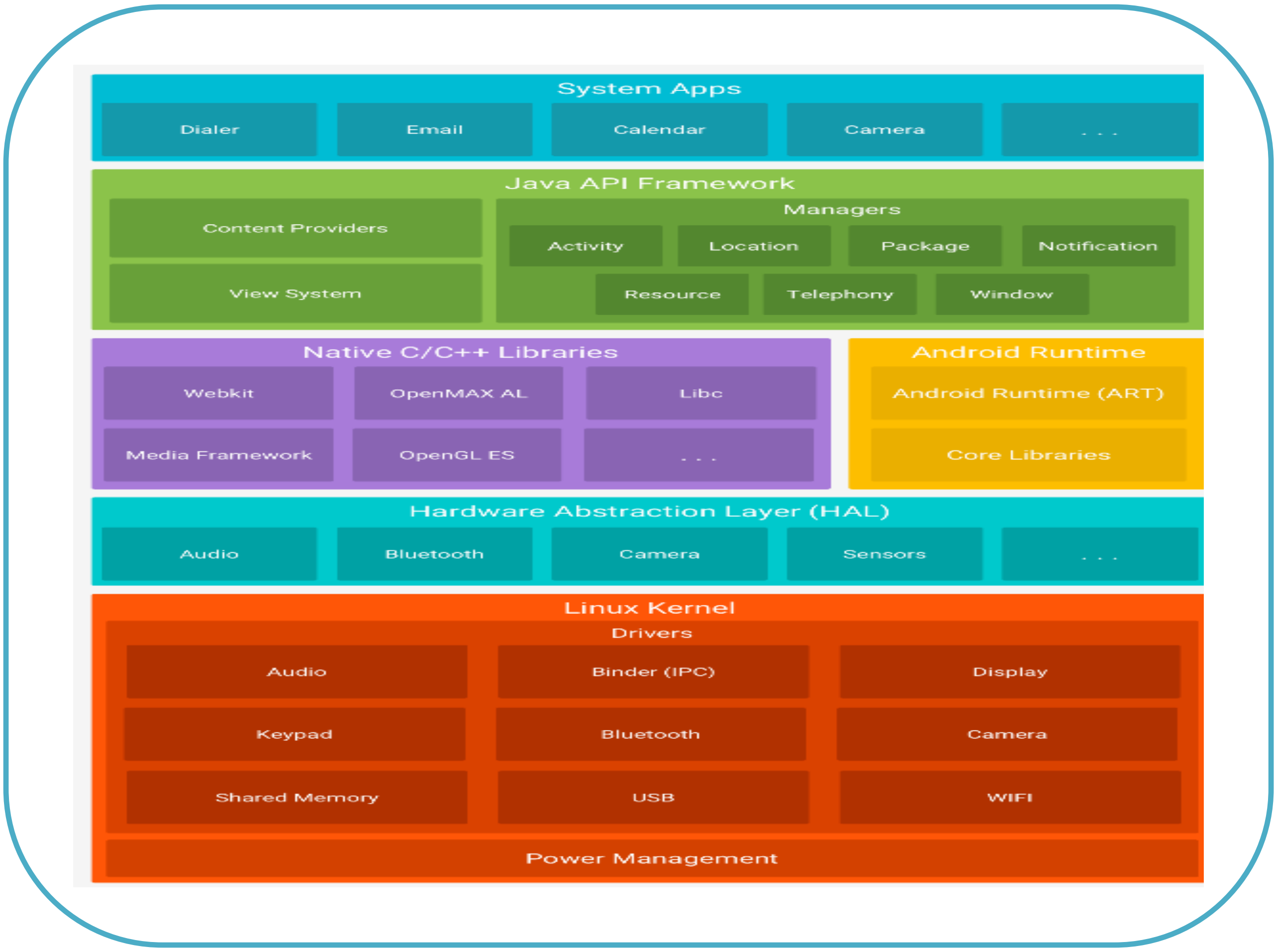

Figure 1. Android Architecture [19]

4.2 The Android Security Model

The security architecture of Android is specifically designed to protect user data and system resources from unauthorized access and malware. The core of this security approach is the inherent security capabilities of the Linux kernel, which are further strengthened by Android-specific features such powerful application sandboxing. Android implements application sandboxing by assigning unique user IDs (UIDs) to each app during installation.[23] This approach guarantees that all processes begun by an application are carried out within a restricted environment, completely separated from other programs. Isolating apps in this manner serves to safeguard against unauthorized access or interference by harmful applications with the functioning of other applications installed on the device. Permissions are essential in Android's security strategy as they regulate access to system resources and sensitive data. Permissions are classified into three groups: normal, risky, and signature. Each group corresponds to a distinct amount of risk and influence on user privacy [22]. Applications must explicitly solicit these permissions, which are subsequently authorized by the user, either at installation or during runtime, based on the level of sensitivity associated with the sought permissions. Digital signatures enhance Android's security by guaranteeing the verification of all applications prior to installation and during upgrades.[24] Applications require a developer's private key signature for authentication, which ensures that the provenance of applications and updates can be verified and unauthorized modifications are prevented.

4.3 The Threat of Android Malware

Although Android has strong security mechanisms, its open nature and widespread usage make it a common target for malware.[25] Malware on Android refers to a range of harmful software specifically created to infect smartphones, pilfer sensitive information, or recruit devices into botnets.[26] Android malware encompasses several types such as spyware, ransomware, and trojans, each specifically crafted to attack distinct vulnerabilities inside the system. Continuous enhancements to the security features of the operating system are necessary due to the ongoing presence and development of Android malware. As the sophistication of malware authors increases, Android must also enhance its security features and detection methods to properly counter new and emerging threats.[27] Gaining a comprehensive understanding of Android's architecture and security model is essential for comprehending the functioning of these risks and implementing the most effective measures to minimize them. This knowledge is vital for safeguarding the integrity and security of Android devices globally.

5. Feature Selection

Feature selection is a fundamental aspect of Android malware detection, as it supports the creation of robust and effective machine learning models. This process is crucial because it involves meticulously analyzing extensive volumes of data to discover and retain the most useful characteristics, which are vital for enhancing the precision and computational efficiency of malware detection systems. Feature selection reduces data complexity, thereby addressing the issue of overfitting in machine learning models. Additionally, it improves the ability of these models to generalize, ensuring good performance on new, and unseen data.

5.1 Importance of Feature Selection

The significant impact of feature selection on the efficacy of detection systems in Android malware detection is well documented. Efficient feature selection methods allow for the reduction of the number of features, which greatly minimizes the processing resources needed. This leads to a faster detection process without sacrificing accuracy. An impressive demonstration of this impact is provided by research conducted by [16], which revealed that incorporating feature selection strategies into machine learning models resulted in a remarkable increase in detection accuracies, from 54.56% to an impressive 74.5%. These improvements underscore the critical role that well-implemented feature selection can play in enhancing the effectiveness of malware detection systems.

The process of feature selection is integral to managing the high-dimensional data commonly encountered in Android malware detection. Without proper feature selection, models may include irrelevant or redundant features, leading to increased computational costs and reduced model performance. [28]By focusing on the most informative features, the model not only becomes more efficient but also more interpretable, allowing for better understanding and trust in its predictions.

Moreover, feature selection is essential in mitigating the curse of dimensionality, a phenomenon where the feature space becomes sparsely populated as the number of features increases, making it difficult for the model to identify meaningful patterns.[29] By reducing the dimensionality, feature selection helps in maintaining a dense distribution of data points, enabling the model to learn more effectively from the available data.

5.2 Feature Selection Techniques

Feature selection in Android malware detection can be categorized into three broad approaches: static, dynamic, and hybrid analysis [30][31]. Each approach offers unique insights into the components of applications, specifically designed to capture particular aspects of their behavior and intrinsic characteristics.

Static Analysis involves examining the code and data of the application without executing it. This includes analyzing manifest files, API calls, and static features like opcode sequences and binary configurations. Static analysis is valued for its efficiency and safety, as it mitigates the risks associated with executing potentially harmful code [32]. It provides a quick method for initial assessments of potential threats within an application. However, its main limitation lies in its inability to detect malware components that are activated dynamically, which can evade detection when analyzed under static conditions. Static analysis relies on the premise that many malicious behaviors are embedded within the application's code and can be identified without execution. For instance, permissions requested by the app, embedded URLs, and cryptographic APIs used can all indicate malicious intent [33]. However, attackers often use obfuscation techniques to disguise these features, making static analysis alone insufficient for comprehensive detection. As a result, static analysis must often be complemented by more dynamic approaches to capture the full spectrum of malicious behaviors.

Dynamic Analysis complements static analysis by observing how an application behaves while running in a controlled environment, such as a sandbox. This approach is crucial for identifying malicious activities that manifest only during the execution of a program, such as runtime permission changes, dynamic library loading, and anomalies in network traffic[34]. Dynamic analysis provides a more comprehensive understanding of an application’s behavior, revealing latent malicious characteristics that static analysis might miss. The primary challenge with dynamic analysis is its resource-intensive nature, requiring substantial computational power and potentially slowing down the detection process. Dynamic analysis captures real-time interactions between the application and the operating system, allowing for the detection of sophisticated attacks that rely on dynamic behaviors, such as trigger-based actions that activate only under certain conditions [30]. This method is particularly effective in identifying malware that uses delayed execution techniques, where malicious actions are initiated only after the application has been running for a certain period or under specific conditions.

However, the complexity of dynamic analysis lies in accurately simulating the real-world environment in which the malware would operate. Malware developers often create environment-aware malware that can detect when it is being run in a sandbox and alter its behavior to avoid detection. [35]To counteract this, dynamic analysis tools must continuously evolve, integrating techniques such as deep learning to identify behavioral patterns indicative of evasion attempts.

Hybrid Analysis seeks to combine the strengths of both static and dynamic analysis into a cohesive approach. By integrating pre-execution inspection with runtime behavioral analysis, hybrid approaches offer a thorough understanding, enhancing detection accuracy and defense against advanced malware attacks [36]. This method is particularly effective in scenarios where both precision and speed of detection are critical, providing a well-rounded solution to the inherent limitations of standalone analysis techniques. Hybrid analysis leverages the efficiency of static analysis for initial threat identification, followed by dynamic analysis to validate and explore potential threats in greater detail. This layered approach ensures that malware, which may pass through static checks undetected, is caught during dynamic execution monitoring [37]. Additionally, hybrid analysis can be further enhanced by integrating network analysis, where traffic patterns are analyzed for signs of malicious communication, such as command-and-control signals. By employing a hybrid approach, security systems can benefit from a more comprehensive dataset that includes both static indicators and dynamic behaviors, leading to more accurate and resilient malware detection [38]. However, the challenge lies in effectively integrating and balancing these two methodologies to avoid overwhelming the system with data and ensuring timely detection and response.

5.3 Specific Feature Selection Methods

Feature selection methods in the context of Android malware detection can be further classified into filter methods, wrapper methods, and embedded methods.

Filter Methods assess the relevance of features based on their statistical properties, such as correlation or mutual information with the target variable [39]. These methods are computationally efficient and can quickly filter out irrelevant features before applying the machine learning model. However, they may overlook interactions between features, which could be important for detecting complex malware behaviors. Filter methods operate independently of any machine learning model, making them versatile and easy to implement. Common techniques include information gain, chi-square tests, and correlation coefficients [40]. These methods are especially useful in the initial stages of feature selection when dealing with large datasets, as they can significantly reduce the feature space, making subsequent model training more manageable. Wrapper Methods involve evaluating feature subsets based on the performance of a specific model. Techniques such as Recursive Feature Elimination (RFE) or stepwise selection are used to iteratively select or remove features and assess the impact on model accuracy [41]. Although more accurate, wrapper methods are computationally expensive and can be impractical for large datasets. Wrapper methods, while resource-intensive, offer a tailored approach to feature selection by directly optimizing model performance.[42] They consider feature interactions and are often used in scenarios where the highest possible model accuracy is desired. However, their computational cost increases exponentially with the number of features, making them less suitable for very high-dimensional datasets unless computational resources are not a constraint.

Embedded Methods perform feature selection during the model training process. For example, LASSO (Least Absolute Shrinkage and Selection Operator) regression can shrink the coefficients of less important features to zero, effectively selecting a subset of relevant feature [43]. These methods are more efficient than wrapper methods and often lead to models with better generalization performance. Embedded methods are integrated within the machine learning algorithm, allowing for feature selection to be part of the model optimization process. Techniques like LASSO and Ridge Regression apply regularization penalties that encourage simplicity and help in managing overfitting by reducing the number of active features [44]. Decision trees and tree-based ensemble methods, such as Random Forests and Gradient Boosting Machines, inherently perform feature selection by identifying the most important features during the training process.

5.4 Challenges in Feature Selection

While feature selection in Android malware detection offers numerous benefits, it also presents several challenges. One prominent issue is the ever-evolving nature of malware. To maintain the effectiveness of detection systems, feature selection algorithms must be regularly updated and adapted to counteract the evolving evasion techniques employed by attackers. The dynamic nature of malware, where new types of attacks are continually developed, poses a significant challenge to static feature selection methods [45]. Features that are indicative of malicious activity today might not be relevant tomorrow, necessitating continuous monitoring and updating of feature selection strategies. This ongoing evolution requires a proactive approach, where new features are identified and incorporated into detection systems before they are widely exploited by attackers.

Moreover, the computational cost associated with feature extraction, especially for dynamic and hybrid analysis methods, poses significant challenges. Achieving a balance between comprehensive analysis and practical computational demands is crucial, particularly for applications requiring real-time detection capabilities [46]. The extraction and processing of features from dynamic analysis, in particular, require significant computational resources. The need to monitor applications in real-time, often across multiple devices and environments, can strain system resources [47]. As a result, optimizing feature selection to minimize computational overhead without sacrificing detection accuracy becomes a critical task. This challenge is amplified in environments with limited processing power, such as mobile devices, where resource efficiency is paramount. Additionally, the importance of features can vary greatly between different types of malware and attack vectors, complicating the process of selecting the most relevant features. Ensuring that the chosen features accurately represent malicious behavior without being overly influenced by benign anomalies or data biases is a persistent challenge.

Feature selection must account for the diversity of malware, which ranges from simple adware to sophisticated ransomware and spyware [48]. Different types of malwares may exhibit different behaviors, making it essential to select features that are general enough to capture a wide range of threats while being specific enough to avoid false positives. Balancing this trade-off requires a deep understanding of the threat landscape and the ability to dynamically adjust feature selection criteria as new threats emerge. Feature selection is an essential component of Android malware detection, playing a vital role in the development of efficient and effective detection systems. By carefully selecting the most relevant features through advanced static, dynamic, and hybrid analysis techniques, and continually adapting to the evolving nature of malware, these systems can maintain robust defenses against a wide array of threats [49]. Continuous refinement and optimization of feature selection strategies are critical to staying ahead in the ever-changing landscape of Android malware. Moving forward, research in feature selection for malware detection should focus on developing adaptive algorithms that can learn and update in real-time, ensuring that feature sets remain relevant despite the rapid evolution of threats. Additionally, integrating more advanced machine learning techniques, such as deep learning, which can automatically learn feature representations from raw data, may further enhance the capability of malware detection systems.[50] Finally, fostering collaboration among researchers to create shared, updated repositories of features and models can help accelerate progress in this field and provide a more unified defense against the ever-growing threat of Android malware.

6. Classification Techniques in Android Malware Detection

Android malware detection has become increasingly dependent on advanced machine learning models, which provide notable benefits compared to conventional approaches. The models, classified as supervised learning, unsupervised learning, and ensemble learning techniques, offer several methodologies to analyze and categorize malware, hence improving the detection capabilities of contemporary security systems [51].

Supervised learning refers to a type of machine learning where a model is trained using labelled data, meaning that the input data is paired with the correct output. This allows the model to learn patterns and make predictions based on new, unseen data. Supervised learning necessitates labelled training data for constructing predictive models. These models acquire the ability to link distinct characteristics of data with predetermined labels, enabling them to categorize novel, unfamiliar data by utilizing acquired patterns. Logistic Regression is a frequently employed model in the field of Android malware detection [52]. Logistic regression models utilize a logistic function to forecast the likelihood of an instance belonging to a specific class. This model utilizes a linear regression to fit an equation to the input features and then performs a logistic transformation to provide a binary output [10]. Logistic regression, although simple and easy to understand, has limitations when it comes to dealing with intricate, non-linear connections between data and outcomes. This can occasionally hinder its capacity to effectively detect advanced malware.
Decision Trees are an often-selected option in the field of supervised learning. These models categorize instances by organizing them using a sequence of decision rules that rely on their feature values. Every node in a decision tree corresponds to a certain feature, while each branch represents a decision rule [18]. Decision trees are used due to their natural and easily understandable structure. However, they frequently experience overfitting, particularly when working with noisy data or data that has a large number of features. Overfitting can result in a high level of accuracy during training, but it often leads to poor generalization when applied to fresh data. Support Vector Machines (SVMs) are highly esteemed as one of the most potent classifiers in the field of Android virus detection. Support Vector Machines (SVMs) function by identifying the most advantageous hyperplane that effectively divides the data into several classes. These models excel in high-dimensional spaces and are frequently employed because of their capacity to handle extensive feature sets [18]. Support Vector Machines (SVMs) have the ability to create both linear and non-linear boundaries, depending on the type of kernel employed. This versatility allows SVMs to effectively handle different types of data. Nevertheless, these algorithms can require significant processing resources, particularly when dealing with extensive datasets, which can present difficulties in real-time detection situations.

Random Forests are a reliable ensemble learning technique that builds many decision trees during training and produces the class that is most frequently predicted by the trees or the average prediction of the individual trees. Random forests mitigate the risk of overfitting, which is commonly associated with individual decision trees, by training numerous decision trees and combining their outputs. This strategy greatly enhances precision and resilience. Research conducted by [53] has demonstrated that random forests can attain remarkable levels of accuracy, occasionally surpassing support vector machines (SVMs). Nevertheless, the intricate nature of random forests can render them less comprehensible in comparison to more straightforward models.

Naive Bayes is a probabilistic classifier that relies on Bayes' theorem and assumes a high level of independence across features. Naive Bayes, despite its straightforwardness, demonstrates effectiveness when applied to extensive datasets and has been utilized in numerous studies focused on detecting Android malware. Ensemble learning approaches benefit greatly from it, as it allows for the combination of predictions from different models to improve overall accuracy [25], [54]. Nevertheless, the concept of independence might be restrictive in practical situations where features frequently exhibit correlation.

The K-Nearest Neighbor (KNN) algorithm is a supervised learning method. The classification is determined by the majority label among the k-nearest neighbors in the feature space [24]. Although KNN can be useful in specific situations, it might be computationally demanding and less efficient when dealing with extensive feature spaces or imbalanced datasets. Due to its substantial memory and processing requirements, it is not well-suited for detecting malware on a broad scale.

6.1 Unsupervised learning

Unsupervised learning models operate without the need for labelled data. Instead, they analyze the incoming data to detect patterns and structures, enabling them to make predictions or group similar instances. This is especially beneficial in situations where there is a limited or nonexistent amount of labelled data. K-Means Clustering is a frequently employed unsupervised learning methodology. This approach utilizes a process of partitioning the data into k clusters, wherein each data point is assigned to the cluster that has the closest mean [21], [23]. K-Means is a commonly employed technique in the field of malware detection to cluster comparable dangerous apps, enabling their collective analysis. Nevertheless, the technique necessitates the pre-specification of the number of clusters and may encounter difficulties when dealing with clusters that have different sizes and densities. Principal Component Analysis (PCA) is a widely used unsupervised learning method for reducing the number of dimensions in a dataset. Principal Component Analysis (PCA) is a statistical technique that converts the data into a collection of orthogonal components, which are arranged in order of the amount of variance they can account for[21], [23]. This strategy is effective in lowering the complexity of the data while preserving a significant amount of the informative variance, hence improving the performance of future classification models.

6.2 Ensemble learning

Ensemble learning methods integrate numerous machine learning models to enhance classification accuracy. Ensemble approaches can enhance the final forecast by combining the results of multiple models, thereby reducing volatility, bias, and improving robustness [55]. Bagging, also known as Bootstrap Aggregating, is a technique that entails training many models using various subsets of the training data and then averaging their predictions [47]. Random forests are a widely used illustration of bagging, a technique in which numerous decision trees are trained using bootstrapped subsets of the data.

Sequential boosting involves training a series of models, with each model aiming to rectify the mistakes made by the previous model. AdaBoost and Gradient Boosting are frequently employed in malware detection to construct robust classifiers by combining multiple weaker models. Boosting can greatly enhance accuracy, but it is susceptible to overfitting if not well controlled. Stacking is the process of teaching a meta-learner to merge the predictions of several base learners, utilizing the advantages of various models to get a more precise ultimate prediction.

6.3 Technique Comparison

Conventional approaches for detecting malware, such as signature-based detection, depend on pre-established rules and patterns to recognize malware [56]. Although these methods are efficient in identifying familiar dangers, they face difficulties in detecting novel or concealed malicious software. On the other hand, machine learning methods have the ability to acquire intricate patterns and apply them to identify malware that has not been encountered before. Supervised learning models such as Support Vector Machines (SVMs) and random forests have shown a high level of accuracy in identifying both familiar and unfamiliar malware by utilizing extensive feature sets derived from Android applications.

Nevertheless, machine learning methodologies also provide difficulties. Supervised learning methods necessitate substantial quantities of annotated data, which can be challenging to acquire [31]. Unsupervised learning models, despite their lack of reliance on labelled input, frequently yield less precise outcomes. Ensemble learning methods, while effective, might require significant computer resources and be challenging to execute.

6.4 Constraints and Difficulties

Implementing machine learning in the field of Android malware detection presents several obstacles. An important problem is overfitting, which occurs when models exhibit high performance on training data but perform badly on unknown data. This is a significant challenge in dynamic contexts where malware undergoes constant evolution [57]. An important difficulty is the presence of data imbalance, where benign samples are more abundant than malicious ones. This results in biassed algorithms that prioritize the majority class. Methods such as data resampling and cost-sensitive learning are frequently used to tackle this problem.

The computing demands of machine learning models, particularly those such as Support Vector Machines (SVMs) and deep learning networks, present substantial obstacles. It is essential to prioritize both prompt detection and precision, especially in real-time applications [58]. The comprehensibility of intricate models such as deep learning networks is likewise restricted, rendering it challenging to grasp and have confidence in their forecasts. The absence of transparency can impede their integration into crucial security systems. To summarize, machine learning techniques provide notable benefits in identifying Android malware compared to conventional approaches. However, they also pose distinct obstacles that require attention. Uninterrupted research and development in this area are crucial to improve the strength, precision, and effectiveness of malware detection systems. By harnessing the capabilities of diverse machine learning models and overcoming their limits, it is feasible to create sophisticated detection systems that can successfully combat the ever-changing threat landscape.

7. Evaluation Metrics and Dataset Considerations in Android Malware Detection

Evaluating the performance of machine learning models in Android malware detection is a critical step to ensure the reliability and effectiveness of the proposed solutions. This section delves into the evaluation metrics commonly used to assess the performance of malware detection models and the datasets typically employed in this field. It also discusses the methodologies for dataset division and the reliability of evaluation results, providing a comprehensive understanding of how to measure and improve detection systems.

7.1 Division of Datasets

A fundamental aspect of evaluating machine learning models is the appropriate division of datasets into training and test sets. This ensures that the model can be trained on one subset of the data and tested on another, providing an unbiased assessment of its performance. Several methods are commonly employed to achieve this division, including hold-out, cross-validation, and bootstrapping.

Hold-out Method: This method involves a straightforward split of the dataset into training and testing sets, typically in a ratio such as 70:30 or 80:20. The training set is used to develop the model, while the test set evaluates its performance [59]. Although simple, this method may not fully utilize all the data for training and can lead to variability in performance metrics depending on the split.

Cross-validation: Particularly k-fold cross-validation, is a more robust method. It involves partitioning the dataset into k mutually exclusive subsets of roughly equal size. The model is trained and tested k times, each time using a different subset as the test set and the remaining subsets as the training set[60]. This process helps ensure that the model's performance is consistent across different splits of the data. The steps are as follows:

1. The dataset is divided into k subsets.
2. One subset is used as the test set, while the remaining k-1 subsets form the training set.
3. This process is repeated k times, with each subset used as the test set once.
4. The performance metrics from the k iterations are averaged to provide an overall evaluation of the model.

Bootstrapping: This technique involves repeatedly sampling the dataset with replacement to create multiple training and test sets [15]. It is particularly useful for estimating the variance of the model's performance and for small datasets where traditional splitting methods might not provide sufficient data for both training and testing.

7.2 Evaluation of Classifier Performance

Evaluating the effectiveness of a classifier in detecting Android malware involves several key performance metrics [61]. These metrics provide insights into different aspects of the classifier's performance, from its overall accuracy to its ability to correctly identify malicious applications without generating too many false positives.

A confusion matrix is a fundamental tool in evaluating classification models. It details the performance of the model by showing the counts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). These counts are used to compute various performance metrics:

Confusion Matrix		
True Class	**Positive**	**Negative**
Positive	TP	FN
Negative	FP	TN

- **Accuracy (Acc):** The ratio of correctly predicted instances (both true positives and true negatives) to the total instances. It provides a general measure of how often the model is correct:

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

- **Error Rate (Err):** The proportion of incorrect predictions (false positives and false negatives) to the total predictions:

  Error Rate= $\frac{FP+FN}{TP+TN+FP+FN}$

- **Precision (P):** The ratio of true positives to the sum of true positives and false positives, reflecting the accuracy of the positive predictions:

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

- **Recall (R):** Also known as sensitivity or true positive rate, recall is the ratio of true positives to the sum of true positives and false negatives. It measures the classifier's ability to correctly identify all relevant instances:

  R = $\frac{TP}{TP+FN}$

  Recall=**F1-Score:** The harmonic mean of precision and recall, providing a single metric that balances both. It is calculated as:

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

- **Area under the ROC Curve (AUC-ROC):** The ROC curve plots the true positive rate against the false positive rate at various threshold settings. The AUC represents the likelihood that the model will rank a randomly chosen positive instance higher than a randomly chosen negative one. A higher AUC indicates better performance, especially in imbalanced datasets where accuracy can be misleading.

7.3 Reliability Estimation of Evaluation Results

To ensure the reliability of the performance metrics, statistical hypothesis testing is often employed. This involves determining whether the observed performance of the classifier is likely to be replicated in real-world scenarios. One common method is the binomial test, which evaluates whether the classifier's error rate on the test set is within an acceptable range. For example, if we want to test the hypothesis that the classifier's generalization error rate is no higher than a threshold ϵ0\epsilon_0ϵ0, we perform the following steps:

1. **Null Hypothesis (H0):** The generalization error rate ϵ is less than or equal to ϵ_0.
2. **Alternative Hypothesis (H1):** The generalization error rate ϵ is greater thanϵ_0.
3. **Significance Level (α):** Commonly set at 0.05, this represents the probability of rejecting the null hypothesis when it is true.
4. **P-value Calculation:** The probability of obtaining test results at least as extreme as the observed results, under the null hypothesis. If the p-value is less than α, the null hypothesis is rejected.

Other statistical tests, such as the T-test, cross-validation T-test, McNemar test, Friedman test, and Nemenyi post-hoc test, can also be used depending on whether we are comparing the performance of one classifier or multiple classifiers.

7.4 Datasets for Android Malware Detection

Obtaining a comprehensive dataset is crucial for training and evaluating malware detection models. Benign applications are often sourced from various Android marketplaces such as Google Play Store, APKMirror, Amazon App Store, and APKPure. However, downloading from these sources does not guarantee benign applications, necessitating the use of tools like VirusTotal to label and verify the dataset.

Benign Datasets: Benign applications are typically sourced from various Android marketplaces. The Google Play Store is the primary source, but APKMirror, Amazon App Store, and APKPure are also commonly used [62]. Additionally, F-Droid, a community-based marketplace, is known for its security, as it only lists open-source applications. Despite this, all downloaded applications should be verified using tools like Virus Total to ensure they are benign.

Malware Datasets: Finding malware datasets is more challenging due to the lower prevalence of malicious applications in official stores. Notable malware datasets include:

- **Drebin Dataset** [13]**:** Contains 5,560 malware applications collected from August 2010 to October 2012. It is frequently used in research due to its comprehensive coverage of malware samples.
- **AMD Dataset** [18]**:** Released in 2017, this dataset includes 24,650 malware applications across 135 families, providing a broad spectrum of malware behaviors.
- **Virus Share** [18]**:** Provides annually updated malicious datasets, which are valuable for tracking the evolution of malware.

These datasets are often augmented with additional malware samples from VirusTotal reports to ensure a comprehensive and up-to-date collection. Ensuring the reliability of the dataset by scanning with VirusTotal helps remove any misclassified applications, thus maintaining the integrity of the dataset.

7.5 Limitations of Current Datasets

Despite the availability of several well-known datasets, such as Drebin and the AMD dataset, there are significant limitations that hinder the effectiveness of malware detection research. One of the primary challenges is the lack of diversity in malware samples. Many datasets tend to over-represent certain types of malwares, which can result in detection models that are biased and less effective against unseen or less common threats.[63] Additionally, the datasets often suffer from an imbalance between benign and malicious samples, leading to models that are more prone to overfitting on the majority class. Moreover, the temporal relevance of these datasets is a critical issue. With the rapid evolution of malware, datasets that are not regularly updated may become outdated, failing to reflect the latest threats and evasion techniques employed by cyber attackers [64]. This issue is exacerbated by the fact that many datasets are static, providing a snapshot of malware from a specific period rather than a continuous update of emerging threats.

7.6 Suggestions for Future Dataset Improvements

To address these limitations, future datasets should aim to encompass a more diverse and representative collection of malware samples [65]. This can be achieved by aggregating data from a wider range of sources, including less commonly monitored app stores, dark web sources, and crowdsourced contributions. Additionally, ensuring a balanced ratio of benign and malicious samples, or providing balanced subsets for training and evaluation, will improve the robustness and generalizability of the resulting detection models. Furthermore, incorporating contextual information, such as the geographical distribution of malware, the types of devices targeted, and the timeframes during which samples were collected, can provide deeper insights, and enhance the model's ability to adapt to different scenarios [4]. Continuous updates and the inclusion of real-time threat intelligence are also essential to keep datasets relevant and reflective of the current threat landscape.

ISSN: 3079-5354

Finally, adopting standardized formats and practices for dataset creation, labeling, and sharing will enhance the reproducibility and comparability of research findings. The creation of open, well-documented datasets that are accessible to the research community can foster collaboration and accelerate advancements in Android malware detection. The evaluation of machine learning models for Android malware detection involves a multi-faceted approach. By employing robust dataset division techniques, utilizing comprehensive performance metrics, and ensuring the reliability of results through statistical testing, researchers can develop and validate effective malware detection systems [64]. The careful selection and use of diverse datasets further ensure that these models are tested against a wide range of real-world scenarios, enhancing their generalizability and robustness. As the threat landscape continues to evolve, ongoing research and refinement of these methodologies remain critical in maintaining effective defense mechanisms against Android malware.

8. Conclusion

The field of Android malware detection has undergone significant transformations, driven by the escalating complexity of threats and the ubiquity of Android devices. The necessity to protect these devices from an ever-growing number of malware variants has led to the integration of advanced machine learning techniques into security systems. This review paper has provided an in-depth exploration of these techniques, emphasizing the critical role they play in modern malware detection.

Initially, this paper highlighted the structural and architectural components of the Android operating system, illustrating how its layered framework, while flexible and versatile, introduces unique security challenges. The intrinsic vulnerabilities within Android's architecture, such as its open-source nature and the widespread use of third-party applications, make it a prime target for malicious attacks. Understanding these foundational aspects is crucial for developing effective detection strategies. We then delved into the various machine learning-based approaches used in Android malware detection, categorized into static, dynamic, and hybrid analysis methods. Static analysis, which examines the application code without execution, provides a foundational layer of defense but falls short in identifying dynamic, runtime threats. On the other hand, dynamic analysis, which observes the behavior of applications during execution, offers deeper insights into potential malicious activities but at the cost of higher computational resources. The hybrid approach, which combines both static and dynamic analysis, presents a more comprehensive solution, capitalizing on the strengths of both methods to enhance detection accuracy.

A significant portion of this review focused on the challenges faced by these detection systems, particularly the deterioration of model performance over time and the sophisticated evasion techniques employed by malware developers. The concept of model deterioration, or concept drift, is a critical issue, as it reflects the inability of static models to cope with the evolving nature of malware. As malware continuously adapts and evolves, models trained on outdated data become less effective, necessitating frequent updates and retraining. This challenge is further compounded by the use of evasion tactics like obfuscation, polymorphism, and dynamic code loading by malware authors, which are specifically designed to bypass traditional detection methods. To mitigate these issues, the review discussed several advanced strategies, such as adversarial training, which involves exposing models to adversarial examples to improve their robustness against evasion. Additionally, the adoption of continuous model updating practices, and the integration of behavioral analysis techniques have been proposed to enhance the adaptability and resilience of detection systems. The importance of feature selection was also underscored in this review. Effective feature selection not only reduces the dimensionality of data but also significantly enhances the accuracy and efficiency of detection models. By focusing on the most relevant features, machine learning models can be trained to detect malware more effectively, with reduced computational overhead and improved generalization capabilities.

Finally, the evaluation of these models was examined through the lens of various performance metrics and the quality of datasets used for training and testing. The review highlighted the limitations of existing datasets, such as biases, lack of diversity in malware samples, and the need for continuous updates to maintain their relevance in the face of rapidly evolving threats. The paper suggested that future research should focus on creating more diverse, balanced, and regularly updated datasets to improve the generalizability of malware detection systems. In conclusion, the battle against Android malware is ongoing, with adversaries constantly innovating to outmaneuver detection systems. While machine learning has brought significant advancements in this field, it is clear that these systems must evolve in tandem with the threats they are designed to counter. Future research must prioritize the development of adaptive, resilient detection models that can withstand the test of time and the ingenuity of malware developers. Collaboration across the research community, combined with the development of robust, open datasets, will be crucial in driving forward the capabilities of Android malware detection and ensuring the security of the vast Android ecosystem.

Corresponding author

Aitizaz Ali
aitizaz.ali@apu.edu.my

Acknowledgements
NA

Funding
NA

Contributions

A.D; J.S; K.P; A.A; Conceptualization, A.D; J.S; K.P; A.A; Investigation, A.D; J.S; K.P; A.A; Writing (Original Draft), A.D; J.S; K.P; A.A; Writing (Review and Editing) Supervision, A.D; J.S; K.P; A.A; Project Administration.

Ethics declarations

This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication

Not applicable.

Competing interests

All authors declare no competing interests.

References

[1] StatCounter. (2024, September 12). *Mobile operating system market share worldwide*. https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202004-202407
[2] Tang, L. (2024, September 12). *Nearly 2,000 victims fell for Android malware scams, at least S$34.1 million lost in 2023*. Channel News Asia. https://www.channelnewsasia.com/singapore/android-malware-scam-millions-lost-cpf-savings-banks-police-4128246
[3] Berghoff, T. (2024, September 12). *G DATA mobile malware report: Harmful Android apps every eight seconds*. G DATA CyberDefense. https://www.gdatasoftware.com/news/1970/01/-36401-g-data-mobile-malware-report-harmful-android-apps-every-eight-seconds
[4] AlJarrah, M. N., Yaseen, Q. M., & Mustafa, A. M. (2022). A context-aware Android malware detection approach using machine learning. *Information (Switzerland), 13*(12). https://doi.org/10.3390/info13120563
[5] Manzil, H. H. R., & Naik, S. M. (2023). Android malware category detection using a novel feature vector-based machine learning model. *Cybersecurity, 6*(1). https://doi.org/10.1186/s42400-023-00139-y
[6] Mahindru, A., & Sangal, A. L. (2021). MLDroid—Framework for Android malware detection using machine learning techniques. *Neural Computing and Applications, 33*(10), 5183–5240. https://doi.org/10.1007/s00521-020-05309-4
[7] Chimeleze, C., et al. (2022). BFEDroid: A feature selection technique to detect malware in Android apps using machine learning. *Security and Communication Networks, 2022*. https://doi.org/10.1155/2022/5339926
[8] Da Costa, L., & Moia, V. (2023). A lightweight and multi-stage approach for Android malware detection using non-invasive machine learning techniques. *IEEE Access, 11*, 73127–73144. https://doi.org/10.1109/ACCESS.2023.3296606
[9] Taher, F., AlFandi, O., Al-kfairy, M., Al Hamadi, H., & Alrabaee, S. (2023). DroidDetectMW: A hybrid intelligent model for Android malware detection. *Applied Sciences (Switzerland), 13*(13). https://doi.org/10.3390/app13137720
[10] Lee, J., Jang, H., Ha, S., & Yoon, Y. (2021). Android malware detection using machine learning with feature selection based on the genetic algorithm. *Mathematics, 9*(21). https://doi.org/10.3390/math9212813
[11] Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2023). A novel permission-based Android malware detection system using feature selection based on linear regression. *Neural Computing and Applications, 35*(7), 4903–4918. https://doi.org/10.1007/s00521-021-05875-1
[12] Roy, S., Bhanja, S., & Das, A. (2023). AndyWar: An intelligent Android malware detection using machine learning. *Innovations in Systems and Software Engineering*. https://doi.org/10.1007/s11334-023-00530-5
[13] Alkahtani, H., & Aldhyani, T. H. H. (2022). Artificial intelligence algorithms for malware detection in Android-operated mobile devices. *Sensors, 22*(6). https://doi.org/10.3390/s22062268

[14] McNeil, P., Shetty, S., Guntu, D., & Barve, G. (2016). SCREDENT: Scalable real-time anomalies detection and notification of targeted malware in mobile devices. *Procedia Computer Science*, *85*, 1219–1225. https://doi.org/10.1016/j.procs.2016.04.254
[15] Smmarwar, S. K., Gupta, G. P., Kumar, S., & Kumar, P. (2022). An optimized and efficient Android malware detection framework for future sustainable computing. *Sustainable Energy Technologies and Assessments*, *54*. https://doi.org/10.1016/j.seta.2022.102852
[16] Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., & Liu, H. (2020). A review of Android malware detection approaches based on machine learning. *IEEE Access*, *8*, 124579–124607. https://doi.org/10.1109/ACCESS.2020.3006143
[17] Senanayake, J., Kalutarage, H., & Al-Kadri, M. O. (2021). Android mobile malware detection using machine learning: A systematic review. *Electronics*, *10*(13), 1606. https://doi.org/10.3390/electronics10131606
[18] Nasri, N. N. M. (2020). Android malware detection system using machine learning. *International Journal of Advanced Trends in Computer Science and Engineering*, *9*(1.5), 327–333. https://doi.org/10.30534/ijatcse/2020/4691.52020
[19] Google Developers. (2024, September 12). *Platform architecture*. Google for Developers. https://developer.android.com/guide/platform
[20] Mansfield-Devine, S. (2012). Android architecture: Attacking the weak points. *Network Security*, *2012*(10). https://doi.org/10.1016/S1353-4858(12)70092-2
[21] Anand. (2017). Android: The architecture and application environment. *International Journal of General Engineering and Technology (IJGET)*, *6*(4).
[22] Sanchez, D., Rojas, A. E., & Florez, H. (2022). Towards a clean architecture for Android apps using model transformations. *IAENG International Journal of Computer Science*, *49*(1).
[23] Mourya, D., Srivastava, S., Pal, D., & Dehraj, P. (2023). A survey: Android architecture and security threats. In *Lecture Notes in Networks and Systems*. https://doi.org/10.1007/978-981-19-4960-9_54
[24] Acharya, S., Rawat, U., & Bhatnagar, R. (2022). A comprehensive review of Android security: Threats, vulnerabilities, malware detection, and analysis. https://doi.org/10.1155/2022/7775917
[25] Lu, T., & Wang, J. (2022). F2DC: Android malware classification based on raw traffic and neural networks. *Computer Networks*, *217*. https://doi.org/10.1016/j.comnet.2022.109320
[26] Kumar, A., Agarwal, V., Shandilya, S. K., Shalaginov, A., Upadhyay, S., & Yadav, B. (2020). PACER: Platform for Android malware classification, performance evaluation and threat reporting. *Future Internet*, *12*(4). https://doi.org/10.3390/FI12040066
[27] Park, M., Seo, J., Han, J., Oh, H., & Lee, K. (2018). Situational awareness framework for threat intelligence measurement of Android malware. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *9*(3). https://doi.org/10.22667/JOWUA.2018.09.30.025
[28] Muzaffar, A., Hassen, H. R., Lones, M. A., & Zantout, H. (2022). An in-depth review of machine learning based Android malware detection. *Computers & Security*. https://doi.org/10.1016/j.cose.2022.102833
[29] Olukoya, O., Mackenzie, L., & Omoronyia, I. (2020). Security-oriented view of app behaviour using textual descriptions and user-granted permission requests. *Computers & Security*, *89*. https://doi.org/10.1016/j.cose.2019.101685
[30] Sugunan, K., Gireesh Kumar, T., & Dhanya, K. A. (2018). Static and dynamic analysis for Android malware detection. In *Advances in Intelligent Systems and Computing*, *645*. https://doi.org/10.1007/978-981-10-7200-0_13
[31] Thangaveloo, R., Jing, W. W., Leng, C. K., & Abdullah, J. (2020). DATDroid: Dynamic analysis technique in Android malware detection. *International Journal of Advanced Science Engineering and Information Technology*, *10*(2). https://doi.org/10.18517/ijaseit.10.2.10238
[32] Ibrahim, M., Issa, B., & Jasser, M. B. (2022). A method for automatic Android malware detection based on static analysis and deep learning. *IEEE Access*, *10*. https://doi.org/10.1109/ACCESS.2022.3219047
[33] Kabakus, A. T. (2019). What static analysis can utmost offer for Android malware detection. *Information Technology and Control*, *48*(2). https://doi.org/10.5755/j01.itc.48.2.21457
[34] Nasser, A. R., Hasan, A. M., & Humaidi, A. J. (2024). DL-AMDet: Deep learning-based malware detector for Android. *Intelligent Systems with Applications*, *21*. https://doi.org/10.1016/j.iswa.2023.200318
[35] Muzaffar, A., Hassen, H. R., Zantout, H., & Lones, M. A. (2023). DroidDissector: A static and dynamic analysis tool for Android malware detection. In *Lecture Notes in Networks and Systems*. https://doi.org/10.1007/978-3-031-40598-3_1
[36] Ding, C., Luktarhan, N., Lu, B., & Zhang, W. (2021). A hybrid analysis-based approach to Android malware family classification. *Entropy*, *23*(8). https://doi.org/10.3390/e23081009
[37] Choudhary, M., & Kishore, B. (2018). HAAMD: Hybrid analysis for Android malware detection. In *2018 International Conference on Computer Communication and Informatics (ICCCI)*. https://doi.org/10.1109/ICCCI.2018.8441295
[38] Hadiprakoso, R. B., Kabetta, H., & Buana, I. K. S. (2020). Hybrid-based malware analysis for effective and efficiency Android malware detection. In *Proceedings of the 2nd International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. https://doi.org/10.1109/ICIMCIS51567.2020.9354315
[39] Smmarwar, S. K., Gupta, G. P., & Kumar, S. (2023). Analysis of feature selection methods for Android malware detection using machine learning techniques. In *Big Data Analytics in Fog-Enabled IoT Networks: Towards a Privacy and Security Perspective*. https://doi.org/10.1201/9781003264545-8
[40] Şahin, D. Ö., Kural, O. E., Akleylek, S., & Kılıç, E. (2023). A novel Android malware detection system: Adaptation of filter-based feature selection methods. *Journal of Ambient Intelligence and Humanized Computing*, *14*(2). https://doi.org/10.1007/s12652-021-03376-6
[41] Guendouz, M., & Amine, A. (2023). A new feature selection method based on Dragonfly algorithm for Android malware detection using machine learning techniques. *International Journal of Information Security and Privacy*, *17*(1). https://doi.org/10.4018/IJISP.319018

[42] Hao, J., Pan, L., Li, R., Yang, P., & Luo, S. (2022). Low redundancy feature selection method for Android malware detection. *Journal of Beijing University of Aeronautics and Astronautics, 48*(2). https://doi.org/10.13700/j.bh.1001-5965.2020.0567
[43] Abawajy, J., Darem, A., & Alhashmi, A. A. (2021). Feature subset selection for malware detection in smart IoT platforms. *Sensors (Switzerland), 21*(4). https://doi.org/10.3390/s21041374
[44] Kshirsagar, D., & Agrawal, P. (2022). A study of feature selection methods for Android malware detection. *Journal of Information and Optimization Sciences, 43*(8). https://doi.org/10.1080/02522667.2022.2133218
[45] Lu, N., Li, D., Shi, W., Vijayakumar, P., Piccialli, F., & Chang, V. (2021). An efficient combined deep neural network based malware detection framework in 5G environment. *Computer Networks, 189*. https://doi.org/10.1016/j.comnet.2021.107932
[46] Fan, M., et al. (2018). Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security*, *13*(8). https://doi.org/10.1109/TIFS.2018.2806891
[47] Ansori, D. B., Slamet, J., Ghufron, M. Z., Putra, M. A. R., & Ahmad, T. (2024). Android malware classification using gain ratio and ensembled machine learning. *International Journal of Safety and Security Engineering*, *14*(1). https://doi.org/10.18280/ijsse.140126
[48] Subbiah, S. S., & Chinnappan, J. (2021). Opportunities and challenges of feature selection methods for high dimensional data: A review. https://doi.org/10.18280/isi.260107
[49] Shen, L., et al. (2023). Self-attention based convolutional-LSTM for Android malware detection using network traffics grayscale image. *Applied Intelligence, 53*(1). https://doi.org/10.1007/s10489-022-03523-2
[50] Keyvanpour, M. R., Shirzad, M. B., & Heydarian, F. (2023). Android malware detection applying feature selection techniques and machine learning. *Multimedia Tools and Applications, 82*(6). https://doi.org/10.1007/s11042-022-13767-2
[51] Guendouz, M., & Amine, A. (2022). A new wrapper-based feature selection technique with Fireworks algorithm for Android malware detection. *International Journal of Software Science and Computational Intelligence, 14*(1). https://doi.org/10.4018/ijssci.312554
[52] Rashed, M., & Suarez-Tangil, G. (2021). An analysis of Android malware classification services. *Sensors*, *21*(16). https://doi.org/10.3390/s21165671
[53] Park, J., Vu, L. N., Bencivengo, G., & Jung, S. (2020). Automatic generation of MAEC and STIX standards for Android malware threat intelligence. *KSII Transactions on Internet and Information Systems, 14*(8). https://doi.org/10.3837/tiis.2020.08.015
[54] Pan, Y., Ge, X., Fang, C., & Fan, Y. (2020). A systematic literature review of Android malware detection using static analysis. *IEEE Access, 8*. https://doi.org/10.1109/ACCESS.2020.3002842
[55] Sumalatha, P., & Mahalakshmi, G. S. (2023). Machine learning based ensemble classifier for Android malware detection. *International Journal of Computer Networks and Communications, 15*(4), 111–122. https://doi.org/10.5121/ijcnc.2023.15407
[56] Yang, J., Tang, J., Yan, R., & Xiang, T. (2022). Android malware detection method based on permission complement and API calls. *Chinese Journal of Electronics*, *31*(4). https://doi.org/10.1049/cje.2020.00.217
[57] Feng, P., Ma, J., Sun, C., Xu, X., & Ma, Y. (2018). A novel dynamic Android malware detection system with ensemble learning. *IEEE Access*, *6*, 30996–31011. https://doi.org/10.1109/ACCESS.2018.2844349
[58] Mohamad Arif, J., Ab Razak, M. F., Awang, S., Tuan Mat, S. R., Ismail, N. S. N., & Firdaus, A. (2021). A static analysis approach for Android permission-based malware detection systems. *PLoS One*, *16*(9). https://doi.org/10.1371/journal.pone.0257968
[59] Sato, R., Chiba, D., & Goto, S. (2013). Detecting Android malware by analyzing manifest files. *Proceedings of the Asia-Pacific Advanced Network, 36*. https://doi.org/10.7125/apan.36.4
[60] Chen, H., Li, Z., Jiang, Q., Rasool, A., & Chen, L. (2021). A hierarchical approach for Android malware detection using authorization-sensitive features. *Electronics (Switzerland), 10*(4), 1–24. https://doi.org/10.3390/electronics10040432
[61] Sharma, R. M., & Agrawal, C. P. (2022). MH-DLdroid: A meta-heuristic and deep learning-based hybrid approach for Android malware detection. *International Journal of Intelligent Engineering and Systems, 15*(4), 425–435. https://doi.org/10.22266/ijies2022.0831.38
[62] Ashawa, M., & Morris, S. (n.d.). *Analysis of Android malware detection techniques: A systematic review*. http://sdiwc.net/digital-library/analysis-of-android-malware-detection-techniques-a-systematic-review
[63] Alzubaidi, A. (n.d.). *Sustainable Android malware detection scheme using deep learning algorithm*. http://amd.arguslab.org/
[64] Akhtar, M. S., & Feng, T. (2022). Malware analysis and detection using machine learning algorithms. *Symmetry*, *14*(11). https://doi.org/10.3390/sym14112304
[65] Meijin, L., et al. (2022). A systematic overview of Android malware detection. *Applied Artificial Intelligence*. https://doi.org/10.1080/08839514.2021.2007327

ISSN: 3079-5354

Biographies

Dr. Aneesha Davarasan, Department School of Technology Networks, Security, Forensic Asia Pacific University of Technology & Innovation (APU), Malaysis. tp063020@mail.apu.edu.my

Joshua Samual, Department School of Technology Networks, Security, Forensic Asia Pacific University of Technology & Innovation (APU), Malaysis. joshua.samual@apu.edu.my

Kulothunkan Palansundram, Department School of Technology Networks, Security, Forensic Asia Pacific University of Technology & Innovation (APU), Malaysis. kulothunkan@apu.edu.my

Dr. Aitizaz Ali received the master's degree in computer systems engineering (with distinction) from GIK Institute, Topi, Khyber Pakhtunkhwa, Pakistan, and the Ph.D. degree in cybersecurity and blockchain technology from the School of IT, Monash University Malaysia, Jaya, Malaysia. He is a Lecturer with the School of IT, UNITAR International University, Petaling Jaya, Malaysia. He is the author of several Journal papers and international Conferences. He has authored or coauthored more than 20 research papers, including in highquality journals. His research interests include blockchain, cloud Ccomputing, cybersecurity, cryptography, deep learning, AI, and healthcare systems. Moreover. He was the Reviewer of IEEE Internet of Things Journal, IEEE Transactions on Network Science and Engineering, IEEE Access, IET, and Human-centric Computing and Information Sciences Journals for several years. aitizaz.ali@apu.edu.my