Available online at www.HighTechJournal.org

HighTech and Innovation Journal

ISSN: 2723-9535

Vol. 5, No. 3, September, 2024

Review Article

A Study of Android Security Vulnerabilities and Their Future Prospects

Albandari Alsumayt [1*], Heba Elbeh [1], Mohamed Elkawkagy [1], Zeyad Alfawaer [2], Fatemah H. Alghamedy [1], Majid Alshammari [3], Sumayh S. Aljameel [4], Sarah Albassam [4], Shahad AlGhareeb [4], Khadijah Alamoudi [4]

[1] *Department of Computer Science, Applied College, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia.*

[2] *Department of Science Technology and Mathematics, College of Art and Sciences, Lincoln University, Jefferson City, MO, United States.*

[3] *Department of Information Technology, College of Computers and Information Technology, Taif University, Taif 21944, Saudi Arabia.*

[4] *Saudi Aramco Cybersecurity Chair, Computer Science Department, College of Computer Science and Information Technology, Imam Abdulrahman Bin Faisal University, Dammam 31441, Saudi Arabia.*

Received 13 April 2024; Revised 25 July 2024; Accepted 07 August 2024; Published 01 September 2024

Abstract

Nowadays, smartphones are used for various activities, including checking emails, paying bills, and playing games, which have become essential parts of daily life. Also, IoT devices can be managed and controlled using applications. While applications can provide numerous benefits, they have also led to several security risks, such as theft of data, eavesdropping, compromised data, and denial-of-service attacks. This study examines security breaches, attacks targeting Android system applications, and vulnerabilities present at every layer of the Android architecture. Additionally, the study aims to compare and evaluate various treatment methods to identify their advantages and disadvantages. Furthermore, the study aims to examine Android's architecture for weaknesses that might lead to app vulnerabilities and potential attacks. To achieve the objectives of this study, a comprehensive analysis of security breaches and attacks targeting Android system applications will be conducted. Various treatment methods will be compared and evaluated through rigorous examination. Additionally, Android's architecture will be thoroughly examined to identify potential weaknesses and vulnerabilities. The analysis will focus on identifying the security risks associated with the use of applications on smartphones and IoT devices. The vulnerabilities present at every layer of the Android architecture will also be analyzed. Furthermore, the advantages and disadvantages of various treatment methods will be assessed. The findings of this study will reveal the various security risks, vulnerabilities, and potential weaknesses present in Android system applications and the Android architecture. The advantages and disadvantages of different treatment methods will also be highlighted. This study contributes to the development of more precise and robust security measures for Android, aiming to mitigate security breaches, attacks, and vulnerabilities. By identifying weaknesses and vulnerabilities, this study provides valuable insights for improving the overall security of Android system applications.

Keywords: DoS; Android; Internet of Things; IoT; Security; Attacks; Detection.

1. Introduction

In the era of digitalization, mobile operating systems have become an integral part of our daily lives. Among them, Android and iOS are the most prevalent, with Android holding a global market share of over 71.74% as of January 2023 [1, 2]. App stores such as Apple's App Store and Google's Play Store have released a total of 4.76 million apps. Each

* Corresponding author: afaalsumayt@iau.edu.sa

http://dx.doi.org/10.28991/HIJ-2024-05-03-020

month, more than 70,000 new Android apps are launched on the Google Play Store, while more than 32,000 new iOS apps are launched on the Apple App Store. The Android mobile operating system is based on Linux, and its source code is available under the Apache License. A license fee is not required for developers to use the Android Software Development Kit, and developers can collaborate with the Android community to incorporate new releases into their apps [3]. The dominance of Android can be attributed to its open-source nature, which allows developers and manufacturers to adapt their designs to meet their needs, leading to faster application development. However, this flexibility also introduces a wide range of security vulnerabilities, which have become a significant concern in the digital world [4, 5].

The Android operating system, being based on Linux, is susceptible to a variety of security vulnerabilities. These vulnerabilities can be found in various layers of the system, including the Linux kernel, applications, and framework [3]. They can arise from coding errors, design flaws, or malicious intent [6-8]. Hackers can exploit these vulnerabilities to launch attacks such as code injection, denial-of-service attacks, collusion, and unauthorized access [4, 9]. Despite the multi-layered security architecture of Android, vulnerabilities can still arise at various levels, leading to several security risks [10, 11].

In benign applications, or in other applications for that matter, vulnerabilities may also occur because of unexpected design flaws or coding errors. As a result of these flaws, the Android operating system can be compromised by attackers. Several security risks are associated with Android phones, including DoS attacks, collusion, malicious code injection, permission escalation, and unauthorized access to applications [12, 13].

Android devices play a significant role in the Internet of Things (IoT) ecosystem, connecting IoT devices to the Internet. However, the security vulnerabilities of both Android and IoT devices can compromise the effectiveness of security measures [14, 15]. Google's attempt to create a modified version of Android, called Android Things, specifically for building IoT-enabled systems, faced significant security challenges and was eventually withdrawn due to these concerns [16].

While there is extensive literature on the security vulnerabilities of Android and the various attacks that can be launched against it, there is a lack of comprehensive studies that analyze these vulnerabilities across all layers of the Android architecture. Furthermore, there is a need for more research on the effectiveness of different detection methods for these attacks and their strengths and weaknesses. Eventually, Android began to replace traditional PCs for certain tasks as it developed into a more comprehensive operating system. While Android was able to connect with peripherals, it could not connect with a wide range of PCs. In its original form, Android Things was intended to connect smartphones to a variety of electronic devices running Android OS and to make smartphones compatible with various electronic devices. As a result of security concerns, Google has since withdrawn from Android Things. The Android Things operating system was designed to be secure, but it faced significant security challenges because it was directly connected to the Google Cloud Platform through the Weave protocol, despite being the first significant IoT platform to incorporate Google's Brillo. In addition, Google had partnerships with Intel and NXP, and they collaborated with other chip makers to enhance security, but these measures weren't enough to address all the security concerns. Since most people are not familiar with the complexities of operating system security, Android Things had a hard time communicating its security features to potential customers. In addition, Android Things' cost was a significant consideration. Intel Edison was the only development board that was supported when Android Things was launched, and it cost $80. Depending on the requirements of the project, development costs may be substantial. Consequently, we will focus on Android security [17].

This study aims to fill these gaps in the literature by answering the following questions:

- Which of the Android architecture weaknesses results in increased vulnerabilities?
- What types of attacks can be made against the Android platform?
- What techniques are used to identify and detect these attacks?
- Which approach to detecting abnormal behaviour on Android devices is better than those described in existing literature?

By providing an overview of the various attacks that can target the Android architecture, discussing the detection methods for these attacks, analysing their strengths and weaknesses, and proposing future research, this study aims to contribute to the ongoing efforts to enhance the security of the Android platform.

This paper is structured as follows. An overview of the various attacks that can target the Android architecture is provided in section 2. In section 3, we discuss the detection methods for these attacks, analysing their strengths and weaknesses and providing an overview of the various approaches available. An analysis of existing detection methods is presented in Section 4 followed by a proposal for future research in Section 5.

2. Literature Review

2.1. Search Strategy

To gather necessary resources related to proposed detection methods of security vulnerabilities of Android, we use Google Scholar and ScienceDirect databases to search for relevant articles by applying the following search syntax:

(“Android” + “DoS” + “vulnerabilities” + “security” + “attacks”)

The results were sorted by relevance, which is limited to 2023-2024 and English papers. The first 100 articles in Google Scholar results were included, and in ScienceDirect results, 131 articles were included. Firstly, we excluded papers that are considered review articles, including comprehensive reviews, surveys, study cases, and systematic reviews. In addition, we considered only indexed Scopus journals, which yielded excluded conference papers, book chapters, doctoral dissertations, and preprint papers. Duplicated articles were removed, too. Totally 58 articles were screened carefully by examining the abstract to identify articles that contained information relevant to the scope of the study. The papers that met the following two criteria were included: (1) focus on the security of Android and (2) propose a detection method. In the end, we identify 14 high-quality research studies as tabulated in Table 1. The process is illustrated in Figure 1.

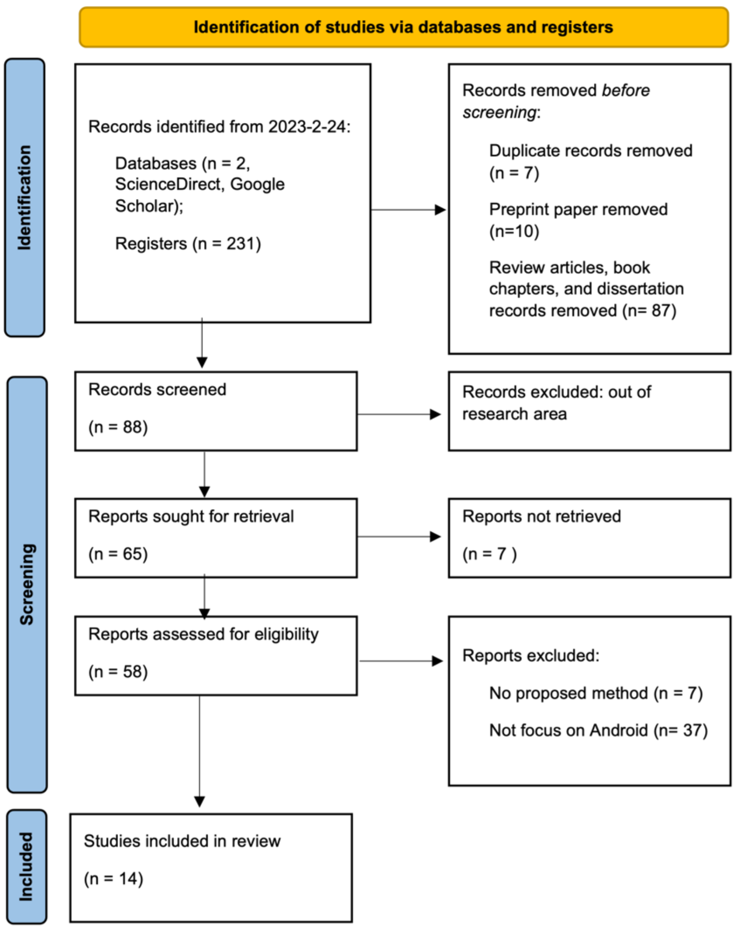
Identification of studies via databases and registers
Identification
Screening
Included
Records identified from 2023-2-24:
Databases (n = 2, ScienceDirect, Google Scholar);
Registers (n = 231)
Records removed *before screening*:
Duplicate records removed (n = 7)
Preprint paper removed (n=10)
Review articles, book chapters, and dissertation records removed (n= 87)
Records screened (n = 88)
Records excluded: out of research area
Reports sought for retrieval (n = 65)
Reports not retrieved (n = 7)
Reports assessed for eligibility (n = 58)
Reports excluded:
No proposed method (n = 7)
Not focus on Android (n= 37)
Studies included in review (n = 14)

Figure 1. PRISMA 2020 flow diagram of the screening and selection procedure

The rapid advancement of technology has brought about significant changes in various sectors, including communication, healthcare, and software development. However, these advancements have also introduced new security challenges. This literature review will focus on recent research in the field of cybersecurity, specifically in areas such as machine learning-based security attack detection, security of inter-app communications, vulnerabilities in Wi-Fi networks, log collection and security analysis in healthcare, attack detection in Android devices, detection of DDoS attacks, security of containers, secure cloud-based mobile apps, operating system vulnerabilities, activity hijacking in Android, and threat detection in smart cyber-physical systems.

Rani et al. [18] introduced a Deep Hierarchical Machine Learning Model (DHMLM) designed to identify multiple security attacks within Device-to-Device (D2D) communication networks, particularly focusing on the challenges posed by 5G/6G technologies. The model aims to address the limitations of traditional Intrusion Detection Systems (IDSs) by offering a hierarchical structure that enhances accuracy, reduces training time, and can identify unknown (Zero-day) attacks. The DHMLM demonstrated superior performance in terms of accuracy, training time, and the ability to identify unknown attacks compared to traditional DNN, RNN, and LSTM approaches. A tool named RONIN was introduced by Romdhana et al. [19] to evaluate the security of Inter-App Communications (ICC) in Android applications. RONIN employs a combination of static analysis, Deep Reinforcement Learning-based dynamic analysis, and software instrumentation to generate exploits for a subset of Android ICC vulnerabilities. The results demonstrated that RONIN surpasses state-of-the-art tools in terms of the number of exploited vulnerabilities. A study conducted in two different universities in Italy evaluated the vulnerabilities in users' configurations that could potentially lead to credential theft [20]. The study revealed that Android devices, in comparison to iOS, provide users with more configuration freedom, making them more susceptible to potential attacks. The authors propose several solutions to address these security vulnerabilities, including the development of a new Extensible Authentication Protocol (EAP) method that does not rely on user passwords, thereby reducing the risk of credential theft. Chidroid (A Mobile Android Application) [21], a novel tool that retrieves, collects, and distributes logs from smart healthcare devices, was introduced. It facilitates the creation of datasets by transforming non-structured data into semi-structured or structured data, which can then be utilized for machine learning and deep learning applications. The application is designed to minimize the impact on system resources and battery consumption. A supervised learning technique was presented that shows promising results in Android malware detection [22]. The approach involves creating a comprehensive labeled dataset of over 18,000 samples, classified into five categories: Adware, Banking, SMS, Riskware, and Benign applications. The model's effectiveness is validated using well-established datasets such as CICMalDroid2020, CICMalDroid2017, and CICAndMal2017.

In Rani et al. [23], the primary goal of the research is to identify and mitigate DDoS and Denial-of-Service (DoS) attacks in D2D communication networks. The researchers created a real-world scenario to simulate Slowloris attacks in a D2D communication network, generating a D2D Network-specific Slowloris dataset. This dataset, along with the CICDDoS2019 dataset, was used to train various Machine Learning (ML) models. The results showed that the Random Forest model provided the best detection. The security of containers in application deployment is a critical concern. Wong et al. [24] used the STRIDE framework for threat modeling and discussed attack analysis and mitigation strategies. The study identified threats such as credential theft, source code tampering, and unauthorized access to host resources. It also reviewed existing mitigation strategies and their limitations, suggesting future research directions. A comprehensive study presented an exhaustive taxonomy of attacks targeting the cloud and mobile ecosystem [25]. The study identified a significant gap in the current approach to software development for cloud and mobile applications. It recommended the adoption of security by design principles and the development of specific frameworks and tools for incorporating security features. An in-depth analysis of vulnerabilities in various operating systems was provided [26]. The study revealed that popular operating systems such as Debian Linux, Android, Windows, and Fedora are highly susceptible to vulnerabilities. The paper concluded by recommending that OS vendors focus on addressing these common weaknesses and advised end-users to stay informed about the latest trends, severity levels, and types of vulnerabilities.

A study addressed the problem of activity hijacking in Android and introduced "*VenomAttack*," an automated and adaptive activity hijacking attack [27]. The paper presented a novel and robust method for conducting activity-hijacking attacks on Android, overcoming the limitations of previous attacks. A novel model designed to protect smart home systems from cyber threats was presented [28]. The model was trained on the IoT Research and Innovation Lab - Smart Home System (IRIL-SHS) testbed dataset. The study concluded that the proposed context-aware threat detection model performs well for smart homes and small offices.

A study introduced a new concept of a Cyber Kill Chain (KC) for IoT devices, known as PETIoT. PETIoT is a novel KC designed to guide penetration testers during Vulnerability Assessment and Penetration Testing (VAPT) sessions over IoT devices [29]. The authors applied PETIoT to a popular IoT device, the TAPO C200 IP camera by TP-Link, to demonstrate its effectiveness. A paper explored the use of AI-based cyber threat detection to safeguard modern digital ecosystems [30]. It evaluated the effectiveness of ML-based classifiers for anomaly-based malware detection and network intrusion detection. The paper suggested that future research should focus on improving the state-of-the-art threat and anomaly detection accuracy to increase the resilience of AI-based defense systems. A research paper discussed the application of Internet of Things (IoT)-based wearable devices for secure lightweight payments in financial technology (FinTech) applications [31]. The authors proposed a novel framework that employs a three-factor authentication system, including biometrics, to ensure secure transactions.

The authors concluded that their proposed framework is secure and efficient for all types of remote and proximity payments using wearable devices. In conclusion, the literature reveals a broad range of cybersecurity threats and vulnerabilities across various domains, including containers, cloud-based mobile apps, operating systems, Android activity hijacking, cyber-physical systems, Wi-Fi enterprise networks, IoT devices, and FinTech applications. The studies also propose various mitigation strategies and future research directions to enhance cybersecurity. Table 1 summarizes the main focus and the key findings or recommendations from each research article:

Table 1. Main findings in recommended papers using Prisma

Title	Main Focus	Key Findings/Recommendations
A Novel Deep Hierarchical Machine Learning Approach for Identification of Known and Unknown Multiple Security Attacks in a D2D Communications Network [18].	offering a hierarchical structure that enhances accuracy, reduces training time, and can identify unknown (Zero-day) attacks	Proposed Deep Hierarchical Machine Learning Model (DHMLM) designed to identify multiple security attacks within Device-to-Device (D2D) communication networks, particularly focusing on the challenges posed by 5G/6G technologies
Assessing the security of inter-app communications in android through reinforcement learning [19].	evaluate the security of Inter-App Communications (ICC) in Android applications	Proposed RONIN, a tool designed to evaluate the security of Inter-App Communications (ICC) in Android applications
Chidroid: A Mobile Android Application for Log Collection and Security Analysis in Healthcare and IoMT [21].	application developed for log collection and security analysis in the healthcare sector	It facilitates the creation of datasets by transforming non-structured data into semi-structured or structured data, which can then be utilized for machine learning and deep learning applications
Deep Learning-Based Attack Detection and Classification in Android Devices [22]	threat of Android malware and the need for robust and efficient solutions for malware detection and classification	present a supervised learning technique that shows promising results in Android malware detection. also developed an Android application that facilitates queries and investigations into the previously studied threats
On the Security of Containers: Threat Modeling, Attack Analysis, and Mitigation Strategies [24].	Security of containers in application deployment	Identified threats include credential theft, source code tampering, and unauthorized access to host resources. Suggested future research directions.
Secure cloud-based mobile apps: attack taxonomy, requirements, mechanisms, tests and automation [25].	Security of cloud-based mobile apps	Identified a significant gap in the current approach to software development for cloud and mobile applications. Recommended the adoption of security by design principles.
Unveiling the Landscape of Operating System Vulnerabilities [26].	Vulnerabilities in various operating systems	Revealed that popular operating systems are highly susceptible to vulnerabilities. Recommended that OS vendors focus on addressing these common weaknesses.
VenomAttack: Automated and Adaptive Activity Hijacking in Android [27].	Activity hijacking in Android	Introduced 'VenomAttack,' a novel method for conducting activity hijacking attacks on Android.
An intelligent context-aware threat detection and response model for smart cyber-physical systems [28].	Threat detection for smart cyber-physical systems	Proposed a novel model for protecting smart home systems from cyber threats. The model performed well for smart homes and small offices.
Attacks and vulnerabilities of Wi-Fi Enterprise networks: User security awareness assessment through credential stealing attack experiments [20].	Security vulnerabilities in Wi-Fi Enterprise networks	Identified vulnerabilities in users' configurations that could potentially lead to credential theft. Suggested designing a new EAP method that does not rely on user passwords.
PETIoT- PEnetration Testing the Internet of Things [29].	Penetration testing for IoT devices	Introduced a new concept of a Cyber Kill Chain for IoT devices, known as PETIoT. Demonstrated its effectiveness on a popular IoT device.
Securing the digital world: Protecting smart infrastructures and digital industries with artificial intelligence (AI)-enabled malware and intrusion detection [30].	AI-based cyber threat detection	Explored the use of AI-based cyber threat detection. Suggested that future research should focus on improving threat and anomaly detection accuracy.
The use of IoT-based wearable devices to ensure secure lightweight payments in FinTech applications [31].	Secure payments in FinTech applications using IoT-based wearable devices	Proposed a novel framework for secure transactions using IoT-based wearable devices. The framework was found to be secure and efficient for all types of payments.

3. Research Methodology

The research questions are answered based on the subsections in the paper which are illustrated in Figure 2.

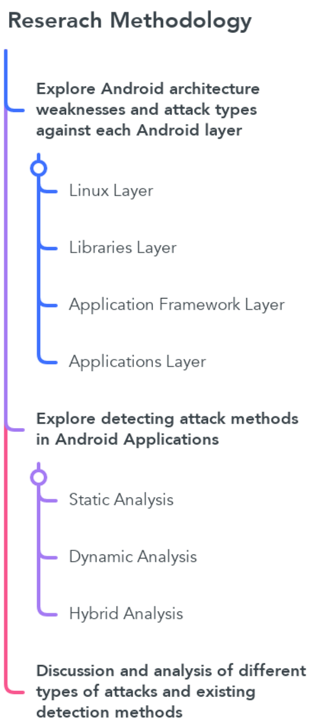
Reserach Methodology
Explore Android architecture weaknesses and attack types against each Android layer
Linux Layer
Libraries Layer
Application Framework Layer
Applications Layer
Explore detecting attack methods in Android Applications
Static Analysis
Dynamic Analysis
Hybrid Analysis
Discussion and analysis of different types of attacks and existing detection methods

Figure 2. Research methodology sections

3.1. Types of Attacks Against Android Systems

In October 2003, Rich Miner, Nick Sears, Andy Rubin, and Chris White founded Android in Palo Alto, California [32]. The project's main objective was to create mobile devices with a high degree of intelligence that could understand the preferences and locations of users. The company's primary motivation during this early stage was to create an advanced operating system incorporating features like digital cameras. As early as the second quarter of 2004, the company had begun to look for investors [33]. As shown in Figure 3, Android OS consists of four main components that are distributed across five layers. Linux kernels, libraries, application frameworks, and applications are among these components [34, 35].

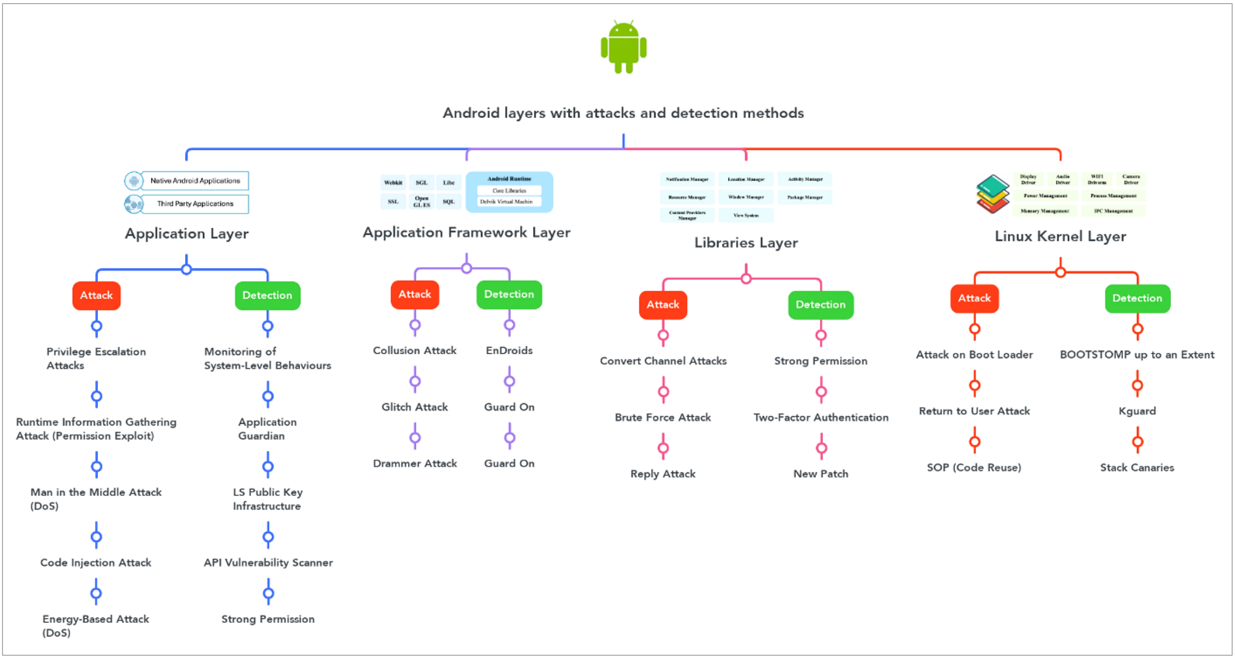

Figure 3. Android layers with attacks and detection methods

3.1.1. Linux Layer

At the bottom of the Android architecture is a layer called the Linux kernel, which plays an important role. This layer provides an abstraction layer between hardware and software and is considered the heart of the Android system. It consists of 115 patches. The Linux kernel also includes hardware drivers such as the display and keypad, as well as device drivers and networking software. The Linux kernel layer manages core system services, including process management, inter-process communication, and physical resource access [36].

At this layer, all Android applications run within a Linux process. A specific Linux user is assigned to each Android application. In this way, Linux's standard access control infrastructure isolates and controls the applications. Through system calls, drivers can access physical resources. System calls are not directly invoked by applications. Instead, the higher-layer services that invoke system calls and claim the necessary services for applications, such as the libraries layer [37]. Different activities and functions are overseen by the kernel in the Android system. Monitoring the Android runtime environment is the responsibility of the Linux kernel. There are, however, vulnerabilities in the kernel and its sections, such as device drivers, runtime environment, and memory, that can be exploited by intruders. Android systems implement a variety of security measures, including secure Inter-Process Communication (IPC), cryptography, an encrypted file system, and the Android Sandbox.

There are several ways to attack this layer, such as targeting a device driver, a boot loader, a memory, or gaining root privileges. This layer can be exploited to gain unauthorized access to the kernel and execute malicious code that reads and writes unauthorized data [38]. These vulnerabilities can be exploited by attackers to execute malicious code. It is also possible for malware to silently infiltrate mobile devices and leak information to the file system and other functionalities, using system resources without the kernel's help [39].

3.1.2. Libraries Layer

In this layer, there are two main modules:

- The first module includes Native C++ libraries, including Webkit, OpenGL, and SSL/TLS, that offer essential advantages to applications. Android system components such as the hardware abstraction layer and Android

Runtime (ART) are written in native code. C or C++ programming languages are typically used to write native libraries for this code. A framework for interacting with the Android system is provided by the Android platform [36].

- Android Runtime (ART) is the second module in this layer, which is a modified Java Virtual Machine (JVM) that runs Android applications that are not natively written. A byte code format designed specifically for Android systems reduces memory consumption. In addition, it contains several virtual machines that are low-memory consuming and can run DEX files. Android's Dalvik Virtual Machine (DVM) is one example of a JVM that is specifically designed for mobile devices. As a result, the device can run multiple instances efficiently, enhance stability, and reduce memory overhead [40].

There is a difference between DVM and JVM [4]. JVM runs Java applications, but Dalvik was specifically designed to optimize the performance of Java apps on devices with limited resources (e.g., low computational power, short battery life, and low memory) [41]. Using DVM, multiple instances can run simultaneously and a variety of features such as memory management, isolation, and threading are available [42]. A separate process in each virtual machine is also provided by DVM as part of its separation feature. Consequently, it does not rely on any other application, so if it crashes, it will not affect other applications. One of the most advanced forms of disruptive attacks is a runtime attack. In order to protect safeguarded assets, conventional security technologies that rely on creating a barrier around safeguarded assets and identifying malicious activity are insufficient [43]. IoT devices with a connection to Android smartphones can rapidly become infected with malware that operates on them. Malicious applications can also gain access to other Internet-connected devices that form part of the Internet of Things [44]. Before the application's runtime, Ahead of Time (AOT) performs extensive bytecode translation in conjunction with ART. Debugging benefits are introduced as well as enhanced garbage collection as a result of this process [45].

3.1.3. Application Framework Layer

Developers can use this layer to integrate various services into their Android applications. There are nine services; firstly, a view system, which offers a comprehensive set of views for creating visually appealing user interfaces for applications. Providers of content facilitate the exchange of data between applications. Resource managers grant permissions to secondary assets such as layouts, strings, and graphics. Resource managers grant permissions to secondary assets such as layouts, strings, and graphics. Every aspect of the application lifecycle and activity is overseen by an activity manager. A location manager determines a user's geographical coordinates. The package manager provides information about the packages available on the device. The creation of screen layouts is enabled by a window manager. There is also a telephony manager that handles network settings on the device [46]. Various attacks can be conducted through this layer, including DoS, privilege escalation, and unauthorized access. Malicious apps could gain unauthorized access to users' data, disable device locks, or use the camera without their consent if a flaw in this layer exists. In this regard, Android should implement up-to-date security measures to ensure that this layer is secure [47]. As a result, malware that runs on Android smartphones can quickly spread to all connected IoT devices. In addition, malicious apps can gain unauthorized access not only to the smartphone but to any IoT devices connected to it as well [48]. In Android applications, developers can utilize a variety of services provided by this layer.

- Android developers can make use of a variety of services in this layer. The nine services are as follows:
- View system: enables applications to be designed visually appealing by using a comprehensive set of views.
- Content provider: Providing content facilitates application-to-application data exchange.
- Resource manager: The resource manager gives permission to secondary assets like layouts, strings, and graphics.
- Notifications manager: allows applications to display alerts and notifications to users.
- Activity manager: An activity manager supervises every aspect of an application's lifecycle and activity.
- Location manager: The location manager determines a user's geographical coordinates.
- Package manager: Information about the available packages on the device is provided by the package manager.
- Window manager: It allows the creation of screen layouts.
- Telephony manager: manages the network settings of the device [39, 46].

Various attacks can be launched using the vulnerabilities in this layer, including DoS attacks, privilege escalations, and unauthorized access. This layer could allow malicious apps to disable device locks, access users' data, or access the device camera without the user's consent. It is therefore vital that Android implements up-to-date security measures for this layer [47]. As an alternative, malware running on Android smartphones can quickly spread to all connected IoT devices. Further, malicious apps can access not only a smartphone but also any connected IoT device.

3.1.4. Applications Layer

In the Android architecture, the Application Layer is the final layer. Android's Application Layer consists of various apps that come pre-installed with the device, including SMS clients, dialers, web browsers, and contact managers [49]. With the Android Application Layer, developers can create new applications and replace the preinstalled ones. IPC is used to share data and functionality between the applications, which run in separate, least-privilege sandboxes [50]. Security- and privacy-sensitive resources, such as location information or contact data, are accessed using middleware and application layer components of the operating system. Permissions are granted by the user to applications for controlling access to sensitive parts of Android. As a result, the Application Layer is susceptible to several types of attacks, such as privilege escalation attacks [8]. Access-granting mechanisms in this layer can be vulnerable to privilege escalation attacks such as Confused Deputy attacks and Collision attacks. Attacks known as Confused Deputy occur when malicious applications misuse other applications to transmit sensitive information, such as contacts, to remote servers. In order to accomplish this, the malicious app must have been granted the READ_CONTACTS permission but not the INTERNET permission. When two malicious applications collide, they often exfiltrate data as a result of working together. Data leakage threats associated with malicious applications can be caused by this type of vulnerability [8, 51].

3.2. Classification of Techniques for Detecting Attacks in Android Applications

Two approaches can be used to identify malicious mobile apps: static analysis and dynamic analysis [52]. In static analysis, the code of the app is deconstructed, including strings, methods, and permissions, and malicious code and manifest files are searched for. In contrast, dynamic analysis detects malicious behavior in a virtualized environment while the app is running [53]. Hybrid analyses can also be performed [54], which combine elements of both methods. Table 2 shows a comparison of detection methods.

Table 2. Comparison of detection methods

Method	Type	Advantages	Disadvantages
Signature-Based Approach	Static	Can result in high processing speed for known programs.	The database needs to be updated regularly, or new malware programs will not be detected.
Permission-Based Analysis	Static	Ensures only necessary resources are allowed for the application to run and with the use of machine learning, the technique can achieve a high level of accuracy.	This technique results in many undetected malware programs and lacks accuracy.
Specification-Based Technique	Static	Can detect both known and unknown instances of malware.	Difficult to specify the behaviour of the system.
Anomaly-Based Detection	Dynamic	Detect unidentified malware.	Significant resource consumption and is not reliable due to false alarms.
Taint Analysis	Dynamic	Track user input and sensitive information flow and locate data leaks.	Cannot track data outside the channel scope.
Emulation-Based Detection	Dynamic	Effective in detecting zero-day attacks.	Malware may detect the virtual environment and circumvent detection. Also, may cause malware infection if the sandbox or the virtual machine is not well configured.
AspectDroid	Hybrid	Can efficiently analyze a diverse set of apps, with very minimal memory and CPU overhead.	Inability to analyze native code.
HADM	Hybrid	The method has high accuracy.	HADM has high computational overhead for large datasets, may not be able to detect new or unknown malware, and may not be effective with real-world data due to scalability.

3.2.1. Static Analysis

Through static analysis, applications are analyzed to identify their features without having to run them on a device or emulator. In the absence of pattern matching, this approach can be challenging to detect malicious behavior. In static analysis, there are three approaches: the signature-based, the permission-based, and the specification-based approaches [55].

Signature-Based Techniques

In the signature-based approach, unique characteristics and patterns are identified in an application and used to create a signature. A malicious program is flagged when its signature matches a known malware signature in the database. Due to the limited number of signatures in the database, this method is commonly used by commercial antimalware products. It is necessary to update the database regularly to detect new malware types [56]. An approach based on behavior for identifying malicious Android app activity has been proposed recently. Application signatures are created based on the behavior of an application, such as leaks of data, jailbreaking, escalation of privileges, and accessing critical permissions during runtime [57].

Permission-Based Techniques

Permission-based detection involves storing the requested permissions in a manifest file. The user must grant permission for the requested resources once the application has been installed. It is possible, however, that some of these resources are not essential for the application to run [58]. In this approach, permissions are examined in the application and verified to be required. There are, however, limitations to this method since it exclusively uses the manifest file as a reference. Some studies suggest combining machine learning with permission-based detection techniques to improve malware detection accuracy [59].

Specification-Based Techniques

A specification-based approach is a variation of anomaly-based detection, which identifies normal, legitimate behavior within an application. With specification-based techniques, predefined rules are used to review programs for malicious activities, and programs that violate these rules are labeled malicious. In contrast, identifying the behavior of the system accurately can be difficult with specification-based techniques [60].

3.2.2. Dynamic Analysis

During its execution, dynamic analysis evaluates a program in real-time. By examining the program's code rather than its code alone, the main goal is to identify errors while the program is in use. One of the key advantages of this approach is that it allows analysis of how the application behaves while it is running. Dynamic analysis requires more resources and therefore takes longer than static analysis. Dynamic analysis includes anomaly detection, taint detection, and emulation-based detection [61].

Anomaly-Based Detection

By training a model to detect unidentified malware, anomaly-based detection identifies programs that exhibit malicious behavior. The model uses features from known malware in order to identify and classify new, unknown malware. A tool that uses this approach offers a thorough analysis, but it requires a significant amount of resources. Detecting malicious applications requires installing the program on the user's device. There is a downside to this method, since it may mistakenly classify legitimate programs as malware if they make a lot of system calls [62].

Taint Analysis

The technique of taint analysis identifies variables that are altered as a result of user input. With TaintDroid, relevant data is tagged with a "taint" and the flow of this tainted data is tracked within the application. A system-wide information flow tracking feature is also available for Android devices. TaintDroid can detect data leaks in third-party applications as well as track various private information sources such as the device's webcam, geolocation, and microphone. This method, however, cannot track data that exits a channel and generates a network response [63].

Emulation-Based Detection

An emulator creates a virtual environment where malware samples can be run to detect malware using an emulation-based detection method. In this way, malware samples are separated from the physical resources of the device, preventing system infection. To prevent infection of other networked devices, it is necessary to configure secure sandboxes and secure virtual machines. The emulation-based detection method is effective at capturing zero-day malware and malware that tries to elevate privileges. In some cases, however, malware is capable of detecting the virtual environment and evading detection [53, 64].

3.3. Hybrid Analysis

Android apps pose an increasing threat to user privacy, which has led to the need for more reliable and accessible analysis techniques. Using hybrid analysis, you can extract all execution paths, even for the most dangerous malware, through analysis of memory dumps and runtime data [65]. AspectDroid and hybrid analysis for malware detection (HADM) are two hybrid analysis techniques.

3.3.1. AspectDroid

The AspectDroid application analyzes Android apps for possible unwanted behaviors and is specifically designed and optimized for Android apps. This solution is flexible and efficient for detecting illicit or suspicious behavior regardless of Android system release or runtime [66]. By leveraging static bytecode tools, AspectDroid weaves analysis routines into existing applications. This allows for efficient detection of resource abuse, data flow analysis, and analytics of suspicious behavior [67].

3.3.2. HADM

The HADM classification method is used to classify Android malware. It converts the information from Android apps into vector-based representations by using 10 static and dynamic features. To determine the best approach, it evaluates the performance of four graph sets and 16 feature vector sets. As part of the method, advanced features derived from deep learning are also incorporated to increase accuracy [68, 69].

4. Discussion and Analysis

In recent years, malicious actors have tried to target and infect Android operating systems with malware due to their widespread use. By analyzing different types of attacks targeting Android systems, researchers can gain valuable insights into the latest trends and vulnerabilities. Depending on the attack, any of the five layers of the Android architecture can be targeted:

- Linux Layer: Linux Layer serves as an intermediary between hardware and software in Android architecture. Often, attackers target its components, including the device driver, runtime environment, and memory, exploiting vulnerabilities in the kernel and its components, including the display and keypad drivers. There are several aspects of this layer that attackers can target, including device drivers, boot loaders, memory, and root privileges. An example of an attack at this layer is unauthorized access to the kernel and the execution of malicious code to read and write memory without permission [70].
- Library Layer: This layer consists of Native C++ libraries and ART, which is a modified JVM. Typically, attackers target vulnerabilities in libraries written in C or C++ to exploit the code in this layer. As a result, they can access sensitive data, modify it, or execute arbitrary code without the user's permission. Android Runtime Environment includes a crucial component known as the DVM, which is a specialized version of the Java Virtual Machine. Malicious code can be executed in the DVM during runtime, allowing attackers to gain unauthorized access to the system or modify data.
- The Application Framework Layer: it provides several services that developers can use in their applications. DoS attacks, privilege escalation, and unauthorized access can be launched through flaws in this layer.
- Applications Layer: The Applications Layer is the top layer of Android architecture. Typical attacks on this layer include stealing sensitive data, tracking user activities, or performing other malicious actions without the user's knowledge. In addition to running on Android smartphones, malware can also be spread to connected IoT devices, allowing attackers to gain unauthorized access to them [71].

Three primary methods of analysis are available to protect against these attacks: dynamic analysis, static analysis, and hybrid analysis. A single approach cannot detect all types of attacks effectively since each method has its advantages and disadvantages. It is therefore possible to achieve a higher detection rate by combining two or more analysis methods.

- A static analysis can be performed in several different ways, including using a signature-based approach, a permission-based approach, and a specification-based approach. By using signature-based techniques, the application's characteristics are compared with a database of known malware signatures. In permission-based techniques, the manifest file is analyzed to ensure the permissions requested by the application are necessary. In specification-based techniques, predefined rules are checked for violations [72].
- A dynamic analysis uses a variety of methods, such as anomaly-based, taint-based, and emulation-based detections. Machine learning algorithms are used to identify patterns indicative of malware using anomaly-based detection. A taint analysis identifies any suspicious or unexpected behavior within the application by tracking the flow of data. Using emulation-based detection, malware samples are executed in a virtual environment to observe their behavior and identify any malicious behavior.
- The hybrid analysis method combines static and dynamic analysis methods. To extract all possible execution paths, it uses techniques such as memory dump analysis and runtime data analysis. There are several types of hybrid analysis techniques, such as AspectDroid and HADM [73].

The Android platform, however, still faces many security challenges and vulnerabilities. To increase the security level in the Android environment, several points should be considered, according to this study. As a first step, users should be vigilant and take measures to secure their systems. You should use strong passwords, avoid phishing sites, and refrain from sharing personal information. Also, any verification requests should be verified to ensure they are coming from an authenticated source. Secondly, application developers should limit the number of permissions needed by their applications to protect the environment. Reduced permissions, for example, can be an effective method of preventing privilege escalation. It is important to grant permissions only for tasks that are required by the application, thereby reducing the number of potential attacks [74]. The Google permission system needs to be strengthened and made more robust from a logical standpoint, which would require an extensive review. As a third step, developers will need to follow established security protocols such as SSL certificates to safeguard their data. Fourth, Android users should only download applications from the Google Play Store, since apps from other sources are not verified and pose significant security risks. Fifth, the use of artificial intelligence and related technologies, such as pattern recognition and machine learning, can contribute to the improvement of security through the detection of anomalies and attacks. Based on peer groups, Google Play groups applications automatically, compares their features, such as permission requests, and uses machine learning to verify applications. In this way, machine learning-based pattern matching on the Google PlayStore can help to detect attacks. Machine learning can enhance security by detecting attacks on the Google PlayStore through pattern matching. This approach may not detect new malware with innovative strategies, such as Kernel Space Mirroring Attacks (KSMAs). The detection of anomalies and gathering of information about these new threats can assist in preventing these threats. Google can use this information to train their systems and detect these threats in the future. In order to develop effective attack detection tools, researchers must prioritize both accuracy and efficiency.

To better understand the effectiveness of different security measures against Android attacks, it is crucial to compare the methodologies adopted in various studies. Each approach targets specific aspects of Android security, utilizing unique strategies to detect, analyze, and mitigate vulnerabilities. The following comparative analysis Table 3 provides a succinct overview of our study, "A Study of Android Security Vulnerabilities and Their Future Prospects," with those of two other significant research efforts: "Assessing Security through Reinforcement Learning" and "Deep Learning-Based Attack Detection," aiming to illustrate the distinct approaches each study takes towards addressing Android security challenges, showcasing how different techniques contribute uniquely to the understanding and mitigation of vulnerabilities in Android systems. This comparison not only highlights the strengths and weaknesses of each method but also illustrates the diverse approaches taken to safeguard Android devices against an evolving landscape of threats.

Table 3. The comparative studies with other studies

Feature	Assessing Security through Reinforcement Learning	Deep Learning-Based Attack Detection	A Study of Android Security Vulnerabilities and Their Future Prospects
Analysis Type	Dynamic	Dynamic	Static and Dynamic
Primary Focus	Exploiting vulnerabilities	Detecting and classifying attacks	Understanding vulnerabilities and attacks; Evaluating treatment methods
Advantages	Can dynamically simulate attacks and test defenses	Real-time behavior analysis for immediate detection	Comprehensive view of system vulnerabilities; Assesses treatment effectiveness
Disadvantages	Computationally intensive	May miss novel attack vectors	Requires extensive data for analysis; May not provide immediate detection
Best Use Scenario	Security testing in lab environments	Real-time system monitoring	Research and development; Policy formulation
Integration with Android	For targeted security testing	Continuous monitoring systems	In-depth analysis and improvement of Android security frameworks

Table 3 presents a comparative analysis of three distinct studies focused on improving the security of Android systems. Each study employs different methodologies to tackle the complex challenges associated with Android security vulnerabilities.

The first study, Assessing Security through Reinforcement Learning [19], utilizes a dynamic approach to actively simulate and test potential security threats in a controlled environment. This method is particularly valuable in a laboratory setting where security systems can be rigorously tested against simulated attacks to identify vulnerabilities before they are exploited in the real world. While this approach offers robust testing capabilities, it requires significant computational resources, making it intensive in terms of time and technology.

The second study, Deep Learning-Based Attack Detection [22], also adopts a dynamic approach but leverages advanced machine learning algorithms to analyze application behaviors in real-time. This method is adept at detecting and classifying patterns that may indicate malicious activities, making it an excellent tool for ongoing system monitoring. The primary advantage of this approach is its ability to provide immediate detection, which is crucial for mitigating fast-acting threats. However, it may not always detect new or unknown attack vectors that haven't been previously learned by the model.

Our study, A Study of Android Security Vulnerabilities and Their Future Prospects, incorporates both static and dynamic analyses to provide a comprehensive overview of vulnerabilities across the Android architecture. This research is critical for understanding the broader security landscape of Android systems, including identifying potential weaknesses that could be targeted by attackers. Your study also evaluates various treatment methods, offering insights into their effectiveness and limitations. This approach is suited for both research and development and policy formulation, aiming to foster a deeper understanding of security solutions and their practical applications in improving Android security.

Together, these studies illustrate the range of techniques available to security researchers and practitioners in combating Android security vulnerabilities. Each approach has its own set of advantages and suitable use scenarios, highlighting the need for a multifaceted strategy when dealing with complex security challenges in Android environments.

5. Conclusion

The Internet has become an indispensable aspect of modern life, with various operating systems and devices catering to this demand. One such operating system is Android, which is built on the Linux kernel and incorporates numerous open-source software components. Android's popularity can be attributed to its user-friendly interface and affordability. In this paper, we have explored the vulnerabilities and attacks that are shared across the Android architecture. Throughout our investigation, we have discussed various detection methods, uncovering potential shortcomings in existing approaches and proposing alternative strategies to address vulnerabilities more effectively in the future. Conducting an extensive survey was imperative to identify the challenges and attacks present within the Android architecture. It is important to note that Android vulnerabilities and detection techniques are constantly evolving, making it necessary to compare multiple detection attacks to understand the advantages and disadvantages of each method.

To ensure the ongoing accuracy of our findings, it is crucial to conduct future surveys that can provide updated information on detection methods and vulnerabilities in Android. By staying up to date with emerging trends, we can adapt our security measures to combat new threats effectively. Additionally, we believe that machine learning can play a significant role in establishing connections between attack patterns and the Android architecture, ultimately enhancing the security of Android systems. In conclusion, this paper has shed light on the vulnerabilities and attacks present within the Android architecture. By exploring various detection methods and highlighting their limitations, we have laid the groundwork for future research to develop more robust solutions. It is our hope that through continued investigation, we can strengthen the security of Android systems, ensuring the safety and privacy of users in the ever-expanding digital landscape.

6. Declarations

6.1. Author Contributions

Conceptualization, A.A. and H.E.; methodology, A.A., M.E., F.H.A.; software, M.A. and Z.A.; validation, S.S.A., S.A., and SH.A.; formal analysis, K.A., A.A., and M.A.; investigation, S.A., H.E., and M.E.; resources, A.A. and Z.E.; data curation, M.A. and F.H.A.; writing—original draft preparation, All authors; writing—review and editing, All authors; visualization, A.A. and F.H.A.; supervision, A.A. and F.H.A.; project administration, A.A. and M.E.; funding acquisition F.H.A. and A.A. All authors have read and agreed to the published version of the manuscript.

6.2. Data Availability Statement

Data sharing is not applicable to this article.

6.3. Funding and Acknowledgment

We would like to thank SAUDI ARAMCO Cybersecurity Chair for funding this article.

6.4. Institutional Review Board Statement

Not applicable.

6.5. Informed Consent Statement

Not applicable.

6.6. Declaration of Competing Interest

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

7. References

[1] Albakri, A., Alhayan, F., Alturki, N., Ahamed, S., & Shamsudheen, S. (2023). Metaheuristics with Deep Learning Model for Cybersecurity and Android Malware Detection and Classification. Applied Sciences (Switzerland), 13(4), 2172. doi:10.3390/app13042172.

[2] Verawati, A., Agustito, D., Pusporini, W., Utami, W. B., & Widodo, S. A. (2022). Designing Android learning media to improve problem-solving skills of ratio. Advances in Mobile Learning Educational Research, 2(1), 216–224. doi:10.25082/amler.2022.01.005.

[3] Wilks, C. R., Gurtovenko, K., Rebmann, K., Williamson, J., Lovell, J., & Wasil, A. R. (2021). A systematic review of dialectical behavior therapy mobile apps for content and usability. Borderline Personality Disorder and Emotion Dysregulation, 8(1), 1–13. doi:10.1186/s40479-021-00167-5.

[4] Mahor, V., Pachlasiya, K., Garg, B., Chouhan, M., Telang, S., & Rawat, R. (2022). Mobile Operating System (Android) Vulnerability Analysis Using Machine Learning. Lecture Notes in Networks and Systems, 481 LNNS, 159–169. doi:10.1007/978-981-19-3182-6_13.

[5] Senanayake, J., Kalutarage, H., Al-Kadri, M. O., Petrovski, A., & Piras, L. (2023). Android Source Code Vulnerability Detection: A Systematic Literature Review. ACM Computing Surveys, 55(9), 1–37. doi:10.1145/3556974.

[6] Saraswat, P. (2023). An inclusive analysis of Google's android operating system and its security. AIP Conference Proceedings, 2427(1), 101614. doi:10.1063/5.0101614.

[7] Sharma, T., & Rattan, D. (2023). Android Malwares with Their Characteristics and Threats. Lecture Notes in Networks and Systems, 588, 1–11. doi:10.1007/978-981-19-7982-8_1.

[8] Yadav, C. S., Singh, J., Yadav, A., Pattanayak, H. S., Kumar, R., Khan, A. A., Haq, M. A., Alhussen, A., & Alharby, S. (2022). Malware Analysis in IoT & Android Systems with Defensive Mechanism. Electronics (Switzerland), 11(15), 2354. doi:10.3390/electronics11152354.

[9] Nouman, N., Noreen, Z., & Naz, F. (2022). Vulnerabilities in Android OS: Challenges and Mitigation Techniques. Lecture Notes in Networks and Systems, 454 LNNS, 256–266. doi:10.1007/978-3-031-01942-5_25.

[10] Ullah, S., Ahmad, T., Buriro, A., Zara, N., & Saha, S. (2022). TrojanDetector: A Multi-Layer Hybrid Approach for Trojan Detection in Android Applications. Applied Sciences (Switzerland), 12(21), 10755. doi:10.3390/app122110755.

[11] Alkahtani, H., & Aldhyani, T. H. H. (2022). Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices. Sensors, 22(6), 2268. doi:10.3390/s22062268.

[12] Roshanaei, M. (2024). Enhancing Mobile Security through Comprehensive Penetration Testing. Journal of Information Security, 15(02), 63–86. doi:10.4236/jis.2024.152006.

[13] Kusreynada, S. U., & Barkah, A. S. (2024). Android Apps Vulnerability Detection with Static and Dynamic Analysis Approach using MOBSF. Journal of Computer Science and Engineering (JCSE), 5(1), 46–63. doi:10.36596/jcse.v5i1.789.

[14] Schiller, E., Aidoo, A., Fuhrer, J., Stahl, J., Ziörjen, M., & Stiller, B. (2022). Landscape of IoT security. Computer Science Review, 44, 100467. doi:10.1016/j.cosrev.2022.100467.

[15] Cirne, A., Sousa, P. R., Resende, J. S., & Antunes, L. (2022). IoT security certifications: Challenges and potential approaches. Computers and Security, 116, 102669. doi:10.1016/j.cose.2022.102669.

[16] Cho, T., Kim, H., & Yi, J. H. (2017). Security Assessment of Code Obfuscation Based on Dynamic Monitoring in Android Things. IEEE Access, 5, 6361–6371. doi:10.1109/ACCESS.2017.2693388.

[17] Novac, O. C., Novac, M., Gordan, C., Berczes, T., & Bujdoso, G. (2017). Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems. 2017 14th International Conference on Engineering of Modern Electric Systems, EMES 2017, 154–159. doi:10.1109/EMES.2017.7980403.

[18] Rani, S. V. J., Ioannou, I. I., Nagaradjane, P., Christophorou, C., Vassiliou, V., Yarramsetti, H., Shridhar, S., Balaji, L. M., & Pitsillides, A. (2023). A Novel Deep Hierarchical Machine Learning Approach for Identification of Known and Unknown Multiple Security Attacks in a D2D Communications Network. IEEE Access, 11, 95161–95194. doi:10.1109/ACCESS.2023.3308036.

[19] Romdhana, A., Merlo, A., Ceccato, M., & Tonella, P. (2023). Assessing the security of inter-app communications in android through reinforcement learning. Computers and Security, 131, 103311. doi:10.1016/j.cose.2023.103311.

[20] Palamà, I., Amici, A., Bellicini, G., Gringoli, F., Pedretti, F., & Bianchi, G. (2023). Attacks and vulnerabilities of Wi-Fi Enterprise networks: User security awareness assessment through credential stealing attack experiments. Computer Communications, 212, 129–140. doi:10.1016/j.comcom.2023.09.031.

[21] Karagiannis, S., Ribeiro, L. L., Ntantogian, C., Magkos, E., & Campos, L. M. (2023). Chidroid: A Mobile Android Application for Log Collection and Security Analysis in Healthcare and IoMT. Applied Sciences (Switzerland), 13(5), 3061. doi:10.3390/app13053061.

[22] Gómez, A., & Muñoz, A. (2023). Deep Learning-Based Attack Detection and Classification in Android Devices. Electronics (Switzerland), 12(15), 3253. doi:10.3390/electronics12153253.

[23] Rani, S. V. J., Ioannou, I., Nagaradjane, P., Christophorou, C., Vassiliou, V., Charan, S., Prakash, S., Parekh, N., & Pitsillides, A. (2023). Detection of DDoS attacks in D2D communications using machine learning approach. Computer Communications, 198, 32–51. doi:10.1016/j.comcom.2022.11.013.

[24] Wong, A. Y., Chekole, E. G., Ochoa, M., & Zhou, J. (2023). On the Security of Containers: Threat Modeling, Attack Analysis, and Mitigation Strategies. Computers and Security, 128, 103140. doi:10.1016/j.cose.2023.103140.

[25] Chimuco, F. T., Sequeiros, J. B. F., Lopes, C. G., Simões, T. M. C., Freire, M. M., & Inácio, P. R. M. (2023). Secure cloud-based mobile apps: attack taxonomy, requirements, mechanisms, tests and automation. International Journal of Information Security, 22(4), 833–867. doi:10.1007/s10207-023-00669-z.

[26] Bhurtel, M., & Rawat, D. B. (2023). Unveiling the Landscape of Operating System Vulnerabilities. Future Internet, 15(7), 248. doi:10.3390/fi15070248.

[27] Sun, P., Chen, S., Fan, L., Gao, P., Song, F., & Yang, M. (2023). VenomAttack: automated and adaptive activity hijacking in Android. Frontiers of Computer Science, 17(1), 171801. doi:10.1007/s11704-021-1126-x.

[28] Noor, Z., Hina, S., Hayat, F., & Shah, G. A. (2023). An intelligent context-aware threat detection and response model for smart cyber-physical systems. Internet of Things (Netherlands), 23, 100843. doi:10.1016/j.iot.2023.100843.

[29] Bella, G., Biondi, P., Bognanni, S., & Esposito, S. (2023). PETIoT: PEnetration Testing the Internet of Things. Internet of Things (Netherlands), 22, 100707. doi:10.1016/j.iot.2023.100707.

[30] Schmitt, M. (2023). Securing the digital world: Protecting smart infrastructures and digital industries with artificial intelligence (AI)-enabled malware and intrusion detection. Journal of Industrial Information Integration, 36, 100520. doi:10.1016/j.jii.2023.100520.

[31] Bojjagani, S., Seelam, N. R., Sharma, N. K., Uyyala, R., Akuri, S. R. C. M., & Maurya, A. K. (2023). The use of IoT-based wearable devices to ensure secure lightweight payments in FinTech applications. Journal of King Saud University - Computer and Information Sciences, 35(9), 101785. doi:10.1016/j.jksuci.2023.101785.

[32] Tripathi, H., Nazir, I., & Singh, S. P. (2021). Android Patient Tracker. EasyChair Preprint, No. 5843, 1-6.

[33] Fajar, A. N., Limonthy, S., Handopo, J. J., Purnawan, F., & Kesuma, A. E. (2023). System Architecture for IT Talent Ecosystem Using Service Oriented Approach. HighTech and Innovation Journal, 4(4), 739-748. doi:10.28991/HIJ-2023-04-04-03.

[34] Sulistyo, W., & Kurniawan, B. (2020). The development of 'JEGER' application using Android platform as history learning media and model. International Journal of Emerging Technologies in Learning (iJET), 15(7), 110-122.

[35] Lopes, J., Serrão, C., Nunes, L., Almeida, A., & Oliveira, J. (2019). Overview of machine learning methods for Android malware identification. 7th International Symposium on Digital Forensics and Security, ISDFS 2019, 1–6. doi:10.1109/ISDFS.2019.8757523.

[36] Sarkar, A., Goyal, A., Hicks, D., Sarkar, D., & Hazra, S. (2019). Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems. Proceedings of the 3rd International Conference on I-SMAC IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2019, 73–79. doi:10.1109/I-SMAC47947.2019.9032440.

[37] Wang, X., & Li, C. (2021). Android malware detection through machine learning on kernel task structures. Neurocomputing, 435, 126–150. doi:10.1016/j.neucom.2020.12.088.

[38] Albakri, A., Fatima, H., Mohammed, M., Ahmed, A., Ali, A., Ali, A., & Elzein, N. M. (2022). Survey on Reverse-Engineering Tools for Android Mobile Devices. Mathematical Problems in Engineering, 2022, 1–7. doi:10.1155/2022/4908134.

[39] Shaheen, J. A., Asghar, M. A., & Hussain, A. (2017). Android OS with its Architecture and Android Application with Dalvik Virtual Machine Review. International Journal of Multimedia and Ubiquitous Engineering, 12(7), 19–30. doi:10.14257/ijmue.2017.12.7.03.

[40] Sutter, T., Kehrer, T., Rennhard, M., Tellenbach, B., & Klein, J. (2024). Dynamic Security Analysis on Android: A Systematic Literature Review. IEEE Access, 12, 57261–57287. doi:10.1109/ACCESS.2024.3390612.

[41] Miltenberger, M., & Arzt, S. (2024). Precisely Extracting Complex Variable Values from Android Apps. ACM Transactions on Software Engineering and Methodology, 33(5). doi:10.1145/3649591.

[42] Zhou, W., Yongzhi, Y., & Wang, J. (2022). Dynamic Class Generating and Loading Technology in Android Web Application. 2022 International Symposium on Networks, Computers and Communications, ISNCC 2022, 1–6. doi:10.1109/ISNCC55209.2022.9851782.

[43] Garg, S., & Baliyan, N. (2024). Mobile OS Vulnerabilities: Quantitative and Qualitative Analysis. CRC Press, Florida, United States. doi:10.1201/9781003354574.

[44] Sharma, T., & Rattan, D. (2021). Malicious application detection in android - A systematic literature review. Computer Science Review, 40, 100373. doi:10.1016/j.cosrev.2021.100373.

[45] Ejiyi, C. J., Deng, J., Ejiyi, T. U., Salako, A. A., Ejiyi, M. B., & Anomihe, C. G. (2021). Design and Development of Android Application for Educational Institutes. Journal of Physics: Conference Series, 1769(1), 12066. doi:10.1088/1742-6596/1769/1/012066.

[46] Dawoud, A., & Bugiel, S. (2021). Bringing Balance to the Force: Dynamic Analysis of the Android Application Framework. 28th Annual Network and Distributed System Security Symposium, NDSS 2021. doi:10.14722/ndss.2021.23106.

[47] Shewale, H., Patil, S., Deshmukh, V., & Singh, P. (2014). Analysis of Android Vulnerabilities and Modern Exploitation Techniques. ICTACT Journal on Communication Technology, 05(01), 863–867. doi:10.21917/ijct.2014.0122.

[48] Razgallah, A., Khoury, R., Hallé, S., & Khanmohammadi, K. (2021). A survey of malware detection in Android apps: Recommendations and perspectives for future research. Computer Science Review, 39, 100358. doi:10.1016/j.cosrev.2020.100358.

[49] Bhat, P., & Dutta, K. (2019). A survey on various threats and current state of security in android platform. ACM Computing Surveys, 52(1), 1–35. doi:10.1145/3301285.

[50] Rana, A. (2021). An overview of android operating system. Academicia: An International Multidisciplinary Research Journal, 11(10), 668-674. doi:10.5958/2249-7137.2021.02115.7.

[51] Heuser, S., Negro, M., Pendyala, P. K., & Sadeghi, A. R. (2017). DroidAuditor: Forensic analysis of application-layer privilege escalation attacks on android (short paper). Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9603 LNCS, 260–268. doi:10.1007/978-3-662-54970-4_15.

[52] Damodaran, A., Troia, F. Di, Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, 13(1), 1–12. doi:10.1007/s11416-015-0261-z.

[53] Ashawa, M., & Morris, S. (2019). Analysis of Android Malware Detection Techniques: A Systematic Review. International Journal of Cyber-Security and Digital Forensics, 8(3), 177–187. doi:10.17781/p002605.

[54] Possemato, A., Nisi, D., & Fratantonio, Y. (2021). Preventing and Detecting State Inference Attacks on Android. In 28th Annual Network and Distributed System Security Symposium, NDSS 2021. doi:10.14722/ndss.2021.24479.

[55] Pan, Y., Ge, X., Fang, C., & Fan, Y. (2020). A Systematic Literature Review of Android Malware Detection Using Static Analysis. IEEE Access, 8, 116363–116379. doi:10.1109/ACCESS.2020.3002842.

[56] Onyedeke, O. C., Elmissaoui, T., Okoronkwo, M. C., Ugwuishiwu, C. H., & Onyebuchi, O. B. (2020). Signature based Network Intrusion Detection System using Feature Selection on Android. International Journal of Advanced Computer Science and Applications, 11(6), 551-558.

[57] Zheng, M., Sun, M., & Lui, J. C. S. (2013). Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware. Proceedings - 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013, 163–171. doi:10.1109/TrustCom.2013.25.

[58] 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. doi:10.1007/s00521-021-05875-1.

[59] Samra, A. A. A., Qunoo, H. N., Al-Rubaie, F., & El-Talli, H. (2019). A survey of static android malware detection techniques. IEEE 7th Palestinian International Conference on Electrical and Computer Engineering, PICECE 2019, 1–6. doi:10.1109/PICECE.2019.8747224.

[60] Dahri, K. A., Vighio, M. S., & Zardari, B. A. (2021). Detection and Prevention of Malware in Android Operating System. Mehran University Research Journal of Engineering and Technology, 40(4), 847–859. doi:10.22581/muet1982.2104.14.

[61] Chao, W., Qun, L., Xiaohu, W., Tianyu, R., Jiahan, D., Guangxin, G., & Enjie, S. (2020). An Android Application Vulnerability Mining Method Based on Static and Dynamic Analysis. Proceedings of 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference, ITOEC 2020, 599–603. doi:10.1109/ITOEC49072.2020.9141575.

[62] Gaharwar, R. S., & Gupta, R. (2020). Android data leakage and anomaly-based Intrusion detection System. 2nd International Conference on Data, Engineering and Applications, IDEA 2020, 1–5. doi:10.1109/IDEA49133.2020.9170738.

[63] Luo, L., Bodden, E., & Spath, J. (2019). A qualitative analysis of android taint-analysis results. Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, 102–114. doi:10.1109/ASE.2019.00020.

[64] Sinha, A., Di Troia, F., Heller, P., & Stamp, M. (2021). Emulation Versus Instrumentation for Android Malware Detection. Advanced Sciences and Technologies for Security Applications, 1–20. doi:10.1007/978-3-030-60425-7_1.

[65] Ding, C., Luktarhan, N., Lu, B., & Zhang, W. (2021). A hybrid analysis-based approach to android malware family classification. Entropy, 23(8), 1009. doi:10.3390/e23081009.

[66] Amin, M., Shah, B., Sharif, A., Ali, T., Kim, K. Il, & Anwar, S. (2022). Android malware detection through generative adversarial networks. Transactions on Emerging Telecommunications Technologies, 33(2), 3675. doi:10.1002/ett.3675.

[67] Ali-Gombe, A. I., Saltaformaggio, B., Ramanujam “Ram,” J. R., Xu, D., & Richard, G. G. (2018). Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. Computers and Security, 73, 235–248. doi:10.1016/j.cose.2017.11.006.

[68] Xu, L., Zhang, D., Jayasena, N., & Cavazos, J. (2018). HADM: Hybrid Analysis for Detection of Malware. Lecture Notes in Networks and Systems, 16, 702–724. doi:10.1007/978-3-319-56991-8_51.

[69] Naway, A., & Li, Y. (2019). Using deep neural network for Android malware detection. arXiv preprint arXiv:1904.00736. doi:10.48550/arXiv.1904.00736.

[70] Roy, R., Dutta, S., Biswas, S., & Banerjee, J. S. (2020). Android things: A comprehensive solution from things to smart display and speaker. Lecture Notes in Networks and Systems, 116, 339–352. doi:10.1007/978-981-15-3020-3_31.

[71] Zhang, Z., Zhang, H., Qian, Z., & Lau, B. (2021). An investigation of the Android kernel patch ecosystem. Proceedings of the 30th USENIX Security Symposium, 3649–3666.

[72] Bhatia, T., & Kaushal, R. (2017). Malware detection in android based on dynamic analysis. 2017 International Conference on Cyber Security and Protection of Digital Services, Cyber Security 2017, 1–6. doi:10.1109/CyberSecPODS.2017.8074847.

[73] Choudhary, M., & Kishore, B. (2018). HAAMD: Hybrid Analysis for Android Malware Detection. 2018 International Conference on Computer Communication and Informatics, ICCCI 2018, 1–4. doi:10.1109/ICCCI.2018.8441295.

[74] Siddiqui, S., & Khan, T. A. (2024). An Overview of Techniques for Obfuscated Android Malware Detection. SN Computer Science, 5(4), 1–24. doi:10.1007/s42979-024-02637-3.