Review

Smartphone Security and Privacy: A Survey on APTs, Sensor-Based Attacks, Side-Channel Attacks, Google Play Attacks, and Defenses

Zia Muhammad [1], Zahid Anwar [1], Abdul Rehman Javed [2], Bilal Saleem [3], Sidra Abbas [4] and Thippa Reddy Gadekallu [2,5,6,7,8,*]

1 Department of Computer Science and the Challey Institute, North Dakota State University, Fargo, ND 6050, USA; zia.muhammad@ndsu.edu (Z.M.); zahid.anwar@ndsu.edu (Z.A.)
2 Department of Electrical and Computer Engineering, Lebanese American University, Byblos 36/S-12, Lebanon
3 Department of Cybersecurity, Air University, E-9, Islamabad 44000, Pakistan; 211096@students@au.edu.pk
4 Department of Computer Science, COMSATS University, Islamabad 44000, Pakistan
5 Zhongda Group, Haiyan County, Jiaxing 314312, China
6 School of Information Technology and Engineering, Vellore Institute of Technology, Tamil Nadu 632 014, India
7 College of Information Science and Engineering, Jiaxing University, Jiaxing 314001, China
8 Division of Research and Development, Lovely Professional University, Phagwara 144411, India
* Correspondence: thippareddy@ieee.org

Abstract: There is an exponential rise in the use of smartphones in government and private institutions due to business dependencies such as communication, virtual meetings, and access to global information. These smartphones are an attractive target for cybercriminals and are one of the leading causes of cyber espionage and sabotage. A large number of sophisticated malware attacks as well as advanced persistent threats (APTs) have been launched on smartphone users. These attacks are becoming significantly more complex, sophisticated, persistent, and undetected for extended periods. Traditionally, devices are targeted by exploiting a vulnerability in the operating system (OS) or device sensors. Nevertheless, there is a rise in APTs, side-channel attacks, sensor-based attacks, and attacks launched through the Google Play Store. Previous research contributions have lacked contemporary threats, and some have proven ineffective against the latest variants of the mobile operating system. In this paper, we conducted an extensive survey of papers over the last 15 years (2009–2023), covering vulnerabilities, contemporary threats, and corresponding defenses. The research highlights APTs, classifies malware variants, defines how sensors are exploited, visualizes multiple ways that side-channel attacks are launched, and provides a comprehensive list of malware families that spread through the Google Play Store. In addition, the research provides details on threat defense solutions, such as malware detection tools and techniques presented in the last decade. Finally, it highlights open issues and identifies the research gap that needs to be addressed to meet the challenges of next-generation smartphones.

Keywords: smartphone security; security and privacy; android issues; malware attacks; APTs; vulnerabilities; sensor-based attacks; side-channel attacks; Google Play Store; Google Play Protect; mobile biometric attacks; static analysis; dynamic analysis; open challenges

check for updates

Citation: Muhammad, Z.; Anwar, Z.; Javed, A.R.; Saleem, B.; Abbas, S.; Gadekallu, T.R. Smartphone Security and Privacy: A Survey on APTs, Sensor-Based Attacks, Side-Channel Attacks, Google Play Attacks, and Defenses. *Technologies* **2023**, *11*, 76. https://doi.org/10.3390/technologies11030076

Academic Editor: Mohammed Mahmoud

Received: 22 May 2023
Revised: 7 June 2023
Accepted: 8 June 2023
Published: 12 June 2023

1. Introduction

The widespread adoption of mobile devices, such as smartphones and tablets, has become increasingly prevalent in recent years. These devices provide individuals with a variety of benefits, including easy access to communication, navigation, entertainment, and social networks [1]. The enormous development and advancement of technology have made mobile devices more affordable, leading to their widespread adoption [2]. The use of mobile devices in offices has also significantly increased due to flexible organizational policies, such as remote network access and bring your own device (BYOD) [3]. According to Zippia statistics, 85% adults in the United States own a smartphone as of the year 2022. The usage of these devices is substantial, with each person spending an average of 5 h and 24 min on their mobile device each day and checking their phones an average of 96 times per day. At the global level, there are approximately 6.65 billion smartphone users. It is recorded that 62.06% of website traffic originates from mobile devices, which shows a significant increase that supersedes all other network devices.

In the current smartphone market, several companies are manufacturing smartphones such as Android, iOS, Windows, Blackberry, Symbian, Tizen, and Harmony operating systems [4–6]. Among all these OS providers, Android-based smartphones are widely adopted and cover up to 87% of the global market share [7]. However, in parallel to increased Android usage, it has become a prime target for cybercriminals, and it is facing a variety of cyber threats. There have been multiple types of malware attacks, such as: Pegasus [8] and SunBird [9]. These are new variants of malware and specifically designed APT that have been launched in users of cellular phones [10]. Most of the time, such kinds of APT are launched to spy and target high profiles, government/ private institutions, surveillance agencies, as well as common users. Mostly, the purpose of these attacks is to spy on target activities by stealing device data, such as call logs, contacts, images, videos, and documents [11,12].

Although researchers are working to advance the understanding of innovation, novelty, and security in mobile manufacturers through user-friendly policies, authentication mechanisms, and security controls [13]. However, Android still needs to address many vulnerabilities that need to be addressed. As the platform continues to evolve, new threats and vulnerabilities are constantly being discovered and reported, making it difficult for researchers to fix the vulnerabilities as quickly as they are discovered. Furthermore, many of the vulnerabilities discovered are not easily fixable and require substantial effort to investigate; as a result, these vulnerabilities often need to be fixed for long periods, leaving Android users vulnerable to malicious attacks. Since the development of Android, it has been identified with approximately 4000 vulnerabilities [14,15].

In recent years, the security landscape for mobile devices has changed dramatically, with attackers increasingly targeting these devices through side-channel attacks [16,17] and malware spread through the Google Play Store [18]. Although previous research has explored vulnerabilities and attacks against mobile devices, these studies often fail to address contemporary threats such as APTs, side-channel attacks, and attacks launched through the Google Play Store. To address this gap in the literature, the underlined article aims to provide a comprehensive overview of the Android ecosystem. It provides a survey of the past 15 years (2009–2023) that highlights security and privacy changes, vulnerabilities, threats, and the corresponding defenses. Additionally, the underlined article provides an in-depth analysis of various types of APTs and malware, including their classification and the methods used to exploit device sensors. Moreover, it also visualizes the various methods used in side-channel attacks [19], including the most common malware families that spread through Google Play.

In addition, the article provides an overview of threat defense solutions, malware detection tools, and techniques that have been developed in recent years. This includes a discussion of the strengths and weaknesses of each approach, as well as an evaluation of their effectiveness. Our research also identifies open issues and highlights the research gap that needs to be addressed to provide robust and effective security for next-generation smartphones. By addressing these issues and identifying areas for future research, we aim to contribute to the development of more secure and reliable mobile devices in the years to come.

1.1. Contributions

1. The survey provides an overview of the current state of smartphone security, identifies trends and patterns in the types of attacks being launched, and illustrates the tactics and techniques employed by attackers.
2. This research will provide the descriptive knowledge required to identify vulnerabilities, APTs, side-channel attacks, and malware families propagated through the Google Play Store. This knowledge will assist mobile manufacturers, anti-malware companies, and Google Play developers in taking preventive measures to ensure the security and privacy of Android users.
3. This research will serve as a valuable resource for the research community to obtain a thorough understanding of the current state of smartphone security and identify areas for future research, open issues, and deficiencies in the Android ecosystem.

1.2. Organization of This Paper

This paper is organized into the following sections. Section 2 provides a comparative study of the existing literature associated with the chosen domain. Section 3 provides the background of the research topic. Section 4 provides details of the survey methodology adopted throughout the article. Section 5 provides a list of vulnerabilities identified in a mobile platform. Section 6 provides details of Android threats. Section 7 provides information on malware that are propagated through the Google Play Store. Section 8 provides details on various possible sensor-based attacks on Android devices. Section 9 provides details of the various side-channel attacks possible on Android smartphones. Section 10 provides details about intrinsic cyberattacks that take advantage of the security vulnerabilities found in the operating system. Section 11 provides details of threat defense and malware detection tools. Section 12 provides details of open issues and solutions. Afterwards, the conclusion and future work has been added in Section 13.

2. Literature Review

The literature review section provides a comprehensive overview of existing research in the field of smartphone security. It serves the purpose of performing the comparative analysis of past efforts, which outlines the research contribution and critically evaluates the research, identifying any gaps or limitations in current knowledge.

Wang et al. [20] conducted a survey of Android malware detection tools and techniques and provided an overview of commonly used algorithms, datasets, evaluation metrics, and indicators. The goal of this article is to provide a guide to Android malware detection and antivirus solutions that can cater to malware spread. Acharya et al. [11] surveyed Android security flaws, primarily focusing on malware detection techniques from 2017 to 2021. This article provides a three-phase model for the identification and characterization of Android malware and provides details of threats and vulnerabilities. Meijin et al. [21] provided an article on a systematic approach to counter software threats and malware attacks. The article provides an overview of Android deterioration problems, malware evasion attacks, and security loopholes using machine learning algorithms [22].

Michael et al. [23] write that the Google Play Store is vulnerable to invasive malware threats. This article provides a detailed study of the security mechanism of the Google Play Store and malware during the period from January 2016 to July 2021. The author identified 107 malware families using control and data flow graphs. Wang et al. [24] presented an article that provides security threats and an overview of the Android security model. The author presents multiple techniques for improving the security infrastructure for developing secure Android firmware, an information encryption module, virus retrieval, and a mobile phone anti-theft module.

Muhammad et al. [25] systematically evaluated Android malware detection tools against contemporary malware. The author surveyed Android malware detection tools and antiviruses based on protection capabilities, accuracy rate, performance, usability, and a tool's ability to classify malware families. Furthermore, the researcher Cheng et al. [26]

investigated multiple attack scenarios or an Android operating system using the smart pentester framework (SPF). The authors exploited multiple Android vulnerabilities using multiple exploitation methods using a virtual environment and Kali Linux. Afterwards, user awareness and attack mitigation techniques were discussed to protect end users from such threats.

A research paper presented by Haowei et al. [27] addressed the sensor leak issue in Android devices and Android wearable hardware. The author worked on hardware and wearable sensors for Android devices, including sensor information leaks and battery-drainage problems. The author developed the SENTINEL tool to test and uncover such leaks in open source and closed source Android applications and smartwatches. Kumar et al. [28] presented a framework for detecting sensor-based threats launched against smart device sensors. The author worked on multiple sensors, including a gyroscope, light, and accelerometer, biometric. These sensors can leak sensitive information and device details to nearby devices and third-party applications. The author presented an intrusion detection system to detect sensor-based attacks and information exposure.

Moreover, Dini et al. [29] presented a user-centric solution against the risks imposed by the Android application. The author surveyed Android applications and proposed a framework entitled the multi-criteria application evaluator of trust for Android (MAETROID). It is designed to ensure the confidentiality and integrity of Android applications based on their risks. Similarly, the author Jalal et al. [30] wrote a survey on mobile security issues, attacks, and vulnerabilities. In addition, a study of the Android security posture was performed. The research covers security aspects and lacks malware threat categorization.

Meng et al. [31] surveyed the Android ecosystem to identify problem areas and potential solutions for secure development. The article provides a detailed list of Android challenges, including issues related to the fragmentation of the ecosystem, the lack of awareness of security among developers, and the complexity of the Android security model. The authors also propose several mitigation strategies to address these challenges, such as the use of best security practices and the adoption of a threat-driven development approach. Similarly, Darell et al. [32] conducted a study on Android security, focusing on identifying prevalent security threats and vulnerabilities. The authors conducted a detailed analysis of existing mobile taxonomy and the current state of the deployment of secure applications in the Android ecosystem. They also evaluated existing security mechanisms and identified gaps in the existing literature.

Parvez et al. [33] reviewed signature-based techniques used by Android malware, such as encryption and code transformation. The review provides an overview of the Android security enforcement mechanisms, threats to these mechanisms, the growth in malware between 2010 and 2014, and the stealth techniques employed by malware authors. Yang et al. [34] focused on the limitations of the current Android permission-based system in preventing malware and proposed a new framework called DroidRisk for assessing the security risk of Android permissions and applications. The article evaluates the effectiveness of DroidRisk using two datasets of benign applications and malware samples and shows that it can generate more reliable risk signals and improve the efficiency of the Android permission system.

Mariantonietta et al. [35] provided an overview of research on security solutions for mobile devices between 2004 and 2011, focusing on high-level attacks targeting user applications. The review categorizes existing approaches to protecting mobile devices into different categories based on detection principles, architectures, collected data, and operating systems, focusing on IDS-based models and tools. This categorization provides an easy and concise view of the underlying models adopted by each approach. Michael et al. [36] surveyed mobile network security, including attack vectors through the back-end system, web browser, hardware layer, and the user's role as an attack enabler. The review highlights the differences and similarities between traditional and mobile security tools.

Alan et al. [37] provided an overview of the security challenges posed by the widespread use of mobile phones in business. The authors surveyed policy and standards, mobile applications, staffing and management, access and authentication, virus and malware, spam, data loss prevention, voice encryption, antitheft, and data backup. The article aims to highlight the challenges posed by mobile phones and determine businesses' readiness to tackle these challenges. The authors also highlight the implications of access to company-confidential information on mobile phones and the use of employees' personal phones for business purposes. Frank et al. [38] provides an overview of the Android operating system and its design differences from a standard Linux operating system. The authors highlight the challenges that embedded mobile devices face and how the Android system was structured to address these challenges. The article notes the use of Linux kernel and various open source libraries in Android and the development of its own C library and Java runtime engine to support higher functionality.

The literature review presented in this article highlights the current state of research in the field of smartphone security, focusing specifically on vulnerabilities and threats to Android devices. The discussed studies contribute to the field by identifying key challenges and proposing solutions. However, as seen in Table 1, previous research contributions lack contemporary threats such as APTs, side-channel attacks, sensor-based attacks, and biometric attacks. Furthermore, some are ineffective against the latest OS variants due to continuous innovation and OS updates.

In contrast, our research aimed to provide a more comprehensive insight into Android security, including an analysis of the security posture, vulnerabilities, cyber threats, and threat defenses. This approach allows for a complete understanding of the current state of Android security and can guide future research in identifying areas for further exploration. Our research aims to build on the existing literature and provide a more comprehensive examination of Android security vulnerabilities, threats, and defenses to counter future-generation smartphone challenges.

Table 1. Comparisons of Previous Works in Relation to the Scope of the Underlined Article.

Authors	**Year**	**Security Posture**	**Vulnerabilities**	**APTs**	**Side-Channel Attacks**	**Sensor-Based Attacks**	**Biometric Threats**	**Play Store Threats**	**Threat Defense**	**Malware Detection**	**Open Challenges**
Frank et al. [38]	2009	✓	×	×	×	×	×	×	×	×	✓
Alan et al. [37]	2010	×	✓	×	×	×	×	×	×	✓	✓
Michael et al. [36]	2011	✓	✓	×	×	×	×	×	✓	×	✓
Mariantonietta et al. [35]	2012	×	✓	×	×	×	×	×	✓	✓	×
Yang et al. [34]	2013	×	✓	×	×	×	×	×	×	✓	✓
Parvez et al. [33]	2014	×	✓	×	×	×	×	×	✓	✓	✓
Darell et al. [32]	2015	✓	✓	×	×	×	×	×	✓	✓	×
Meng et al. [31]	2016	✓	✓	×	×	×	×	×	×	✓	×
Jalal et al. [30]	2017	✓	✓	×	×	×	×	×	✓	×	×
Dini et al. [29]	2018	✓	×	×	×	×	×	×	✓	×	×
Kumar et al. [28]	2019	×	×	×	✓	✓	×	×	✓	✓	×
Haowei et al. [27]	2020	×	×	×	✓	✓	✓	×	×	×	×
Cheng et al. [26]	2021	×	✓	×	✓	✓	×	×	×	×	✓
Muhammad et al. [25]	2021	✓	✓	×	×	×	×	×	✓	✓	✓
Wang et al. [24]	2021	×	✓	×	×	×	×	×	✓	×	×
Michael et al. [23]	2022	✓	×	×	×	×	×	✓	✓	✓	✓
Meijin et al. [21]	2022	✓	×	×	×	×	×	×	✓	✓	×
Acharya et al. [11]	2022	✓	✓	×	×	×	×	×	✓	✓	×
Wang et al. [20]	2023	×	×	✓	×	×	×	×	✓	✓	✓
This Paper	2023	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

3. Background

The background section provides the reader with the necessary information to understand the research's context and significance. This section will help the reader understand the research problem, the knowledge gap that this study aims to fill, and the importance of working in the Android smartphone domain. This section provides a clear and concise explanation of previous research studies related to the current study and sets the stage for the rest of the article.

3.1. Android Open Source Project (AOSP)

Android is an open source Linux-based operating system (OS) that allows users to manage activities of daily living (ADL) [39]. Android OS is based on security-enhanced Linux (SE-Linux) that emphasizes security controls in the Android infrastructure by maintaining a sophisticated set of rules, Linux policies, and a kernel-level application sandboxing mechanism [40]. Additionally, SE-Linux assigns identification numbers (ID) to users and groups that identify and limit users, processes, and application access to kernel level and hardware attributes [32]. In an ideal scenario, an application works in a dedicated sandbox and cannot interact with other applications and resources until evidenced by a predefined set of permissions or a specific need [41].

Android is not just limited to smartphones; it has an entire ecosystem. Android ecosystem refers to the collection of hardware, software, and services built around the Android operating system (OS) [42,43]. Many manufacturers use Android OS to power various devices, including smartphones, tablets, smartwatches, televisions, and vehicle infotainment systems [44,45]. These devices come in various sizes, shapes, and capabilities and are manufactured by many different manufacturers [46,47]. Android-based devices are used in several industries, such as retail, healthcare, transportation, and manufacturing [48]. For a better understanding of the Android ecosystem, Figure 1 provides a detailed overview of the list of devices powered by the Android operating system. The figure helps the reader understand the wide range of devices powered by the Android operating system in all major manufacturing industries.

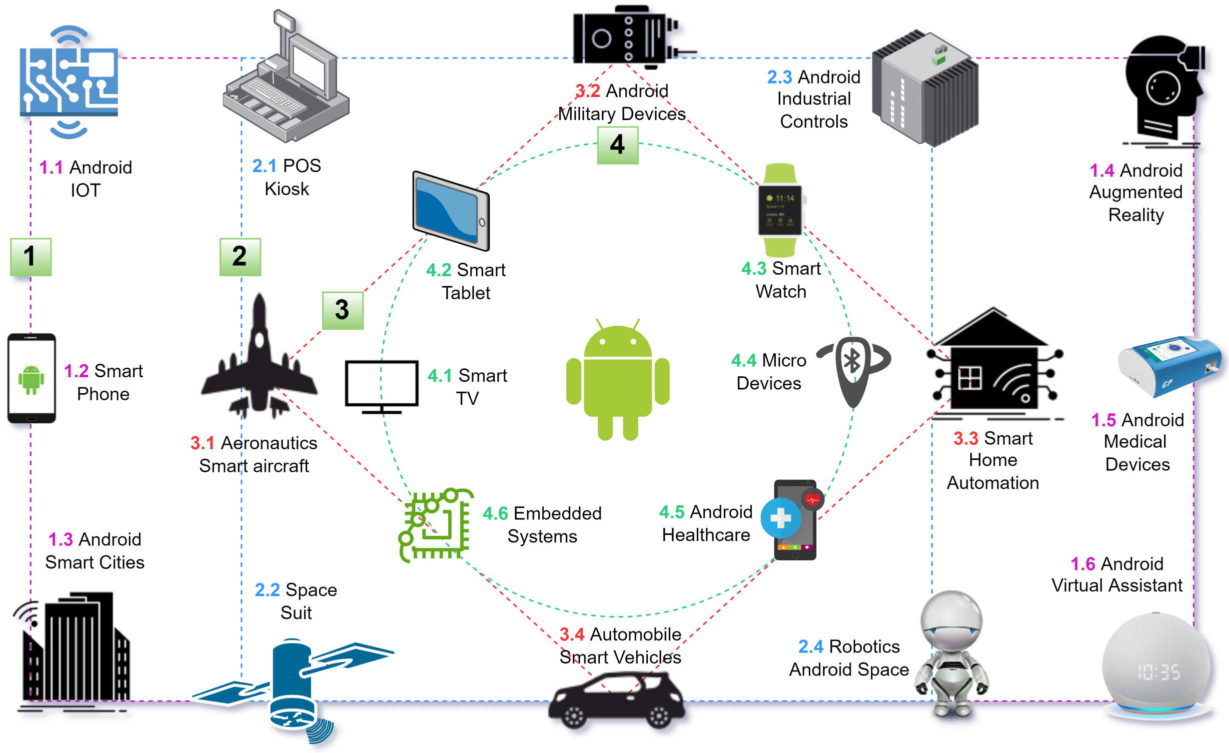

1.1 Android IOT
2.1 POS Kiosk
3.2 Android Military Devices
4
2.3 Android Industrial Controls
1.4 Android Augmented Reality
1
2
3
4.2 Smart Tablet
4.3 Smart Watch
1.2 Smart Phone
4.1 Smart TV
4.4 Micro Devices
3.3 Smart Home Automation
1.5 Android Medical Devices
3.1 Aeronautics Smart aircraft
4.6 Embedded Systems
4.5 Android Healthcare
1.3 Android Smart Cities
2.2 Space Suit
3.4 Automobile Smart Vehicles
2.4 Robotics Android Space
1.6 Android Virtual Assistant
10:35

Figure 1. Android ecosystem: list of global devices that are powered by the Android operating system.

At the border level, Android smartphones are secured by the 3Tier security model; (1) security measures provided by SE-Linux [49]; (2) security measures provided by Google; and (3) security measures provided by the original equipment manufacturer (OEM).

First, *SE-Linux* has policy enforcement mechanisms that ensure security at the root level by providing OS-level controls, access vector cache, object-based security, SE-Linux enforcement mode, mandatory access controls, user ID, and process ID [50].

Second, *Google's security measures* that include Play Protect, Google safety checks, permission declaration, play console, security alerts, and device security update releases [51,51]. Google security measures also include a safety network (SafetyNet) for intrusion detection, tampering detection, and privacy preservation in mobile devices built on top of SE-Linux [52,53].

Third, *OEM security measures* that include security patches, update management, antivirus application, application runtime check, secure vaults, and tracking systems specifically designed by the manufacturer in addition to the security measures of SE-Linux and Google [54].

3.2. Dissecting Android Architecture

Android architecture is a layered architecture composed of five main components: the application framework, interprocess communication (IPC), system services, hardware abstraction layer (HAL), and the Linux kernel. For a thorough understanding, the illustration of the Android architecture is presented in Figure 2. The figure provides a detailed overview of the different components that stack on top of each other, with the Linux kernel at the bottom, followed by the hardware extraction layer, system services, interprocess communication, and the application framework at the top [55]. This design allows for a separation of duties and helps to ensure security and privacy at the application level.

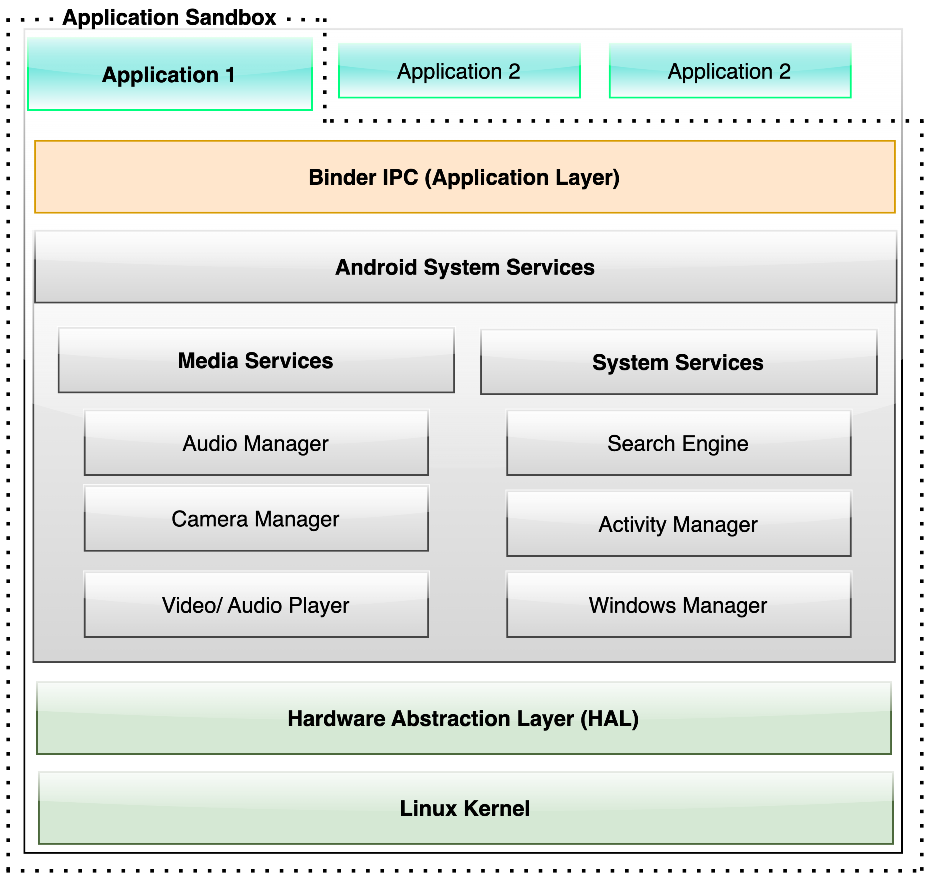


Figure 2. Illustration of an Android architecture.

1. **Application framework:** An application framework is a high-level API for developers to create Android applications. It includes services required for application development, such as an activity manager, content provider, and other libraries. The framework creates and manages the application lifecycle, data storage, information retrieval, and user interface management [56,57].
2. **Interprocess communication (IPC):** IPC allows different applications and services to communicate with each other. It also enables the components of an application to share data, access device functionality, and integrate with third-party services.
3. **System services:** These services handle common functionalities such as power management, location, and connectivity.
4. **Hardware abstraction layer (HAL):** It is an abstraction layer between the Android software and the hardware of the device that allows the Android software to be hardware-agnostic, which means that the same software can run on different devices with different hardware configurations.
5. **Linux kernel:** kernel is responsible for several low-level services such as memory management, process management, and device drivers.

3.3. Google Play Store

Google Play Store, also known as Google Play [58], is an online platform developed by Google that offers a large collection of resources, such as applications, games, movies, television shows, and books. It has more than two million premium and free applications that provide users with an extensive range of options. The Play Store is pre-installed on most Android devices, serving as the default platform for downloading and updating apps. Google advertises the Play Store as one of the most secure platforms for distributing applications, ensuring users' safety and security while downloading and installing apps. The Google Play Store is widely accessible in over 150 countries and languages.

3.4. Google Play Protect

Google Play Protect is a security feature developed by Google to help protect users from malicious apps and other security threats on Android devices [51]. It is built into the Google Play Store and runs automatically in the background on Android devices. One of the key features of Google Play Protect is its ability to scan millions of Android devices for potentially harmful applications (PHAs) and remove them from the Play Store. It also scans the applications downloaded from other sources, such as third-party app stores [59]. If an app is found malicious, Google Play Protect will warn the user and block the app from being installed or running.

3.5. Android Permission Model

The Android permission model is a system that controls access to sensitive resources and data on Android devices. It is designed to protect users' security and privacy by requiring apps to request permission to access certain resources or data on a device [60,61]. The Android permission model is based on the idea that applications should only have access to the resources and data they need to perform their intended functions. For example, an app designed to take photos should request permission to access the camera but should not have access to the user's contacts or location data [62,63].

When an app is installed on an Android device, it must request permission to access specific resources or data. The user is then given a list of the requested permissions and can grant or deny them. The application can access the corresponding resource or data if the user grants permission. The app cannot access the corresponding resource or data if the user denies permission. The Android permission model includes four different categories of permissions. Details of all these categories can be seen in the following paragraphs, and a quick overview is presented in Figure 3. The figure illustrates how the Android permission model is designed to give users control over which resources and data apps can access on their devices while ensuring that apps can perform their intended functions.

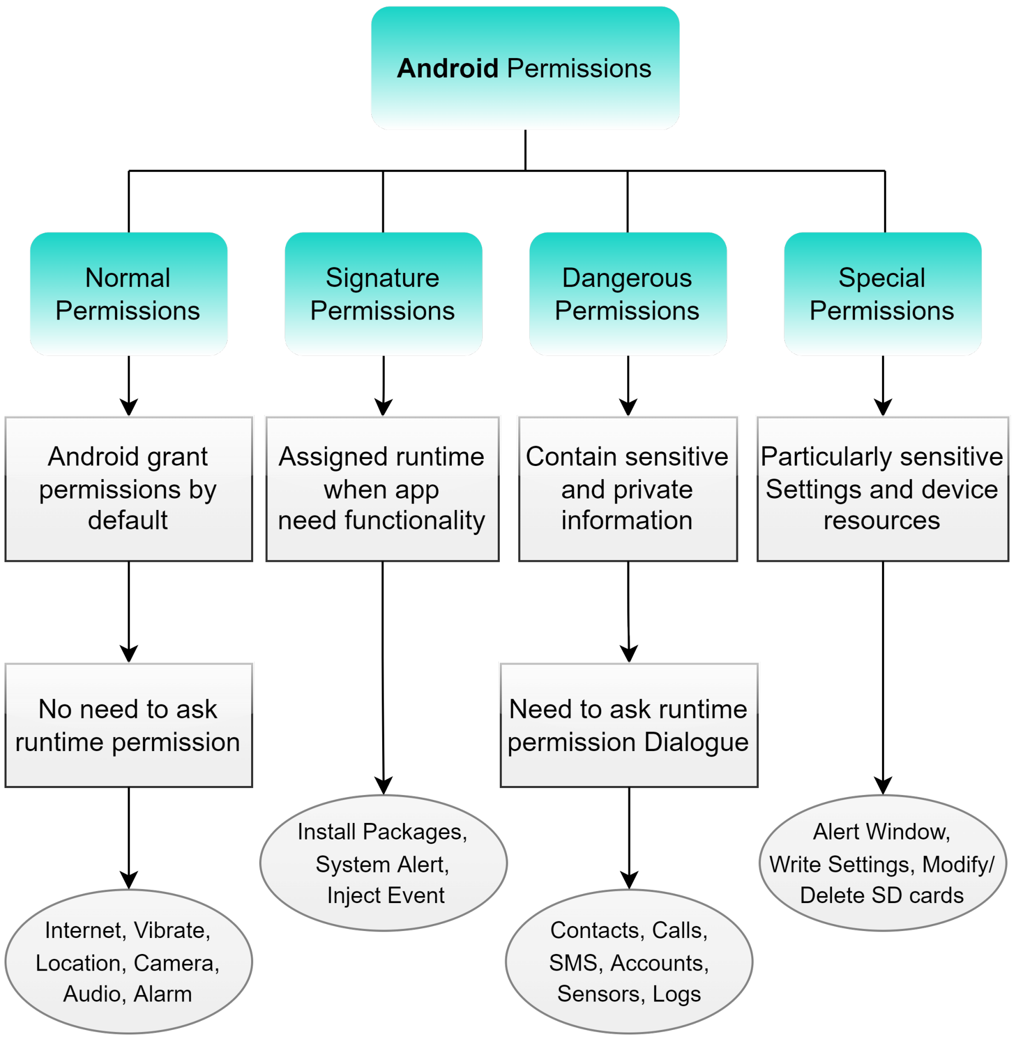

Android Permissions
Normal Permissions
Signature Permissions
Dangerous Permissions
Special Permissions
Android grant permissions by default
Assigned runtime when app need functionality
Contain sensitive and private information
Particularly sensitive Settings and device resources
No need to ask runtime permission
Need to ask runtime permission Dialogue
Internet, Vibrate, Location, Camera, Audio, Alarm
Install Packages, System Alert, Inject Event
Contacts, Calls, SMS, Accounts, Sensors, Logs
Alert Window, Write Settings, Modify/ Delete SD cards

Figure 3. Categorization of the Android permission model.

1. **Normal permissions:** Normal permissions are considered safe and do not pose a significant risk to the user's security or privacy. Examples include permissions to access the Internet, vibrate the device, or write to external storage. These permissions are automatically granted at the installation time and cannot be revoked by the user. App developers do not need to explicitly ask for normal permissions and are granted as soon as the app is installed.
2. **Signature permissions:** Signature permissions are granted at runtime based on the certificate that defines the permission and the application. These permissions are intended for use by applications that are signed with the same certificate and are meant to be used for sharing the functionality between applications. For example, an application signed by the same certificate as the system can use the permission to read the state of the phone, while an application signed by a different certificate cannot.
3. **Dangerous permissions:** Dangerous permissions pose a greater risk to user security or privacy. Examples include permissions to access the camera, microphone, or location data [64,65]. When an app requests dangerous permission, the user is presented with a dialog box explaining why the app needs permission, and the user must explicitly grant or deny the permission. Dangerous permissions must be asked for during runtime, and data can only be accessed if the user agrees to allow permission.
4. **Special permissions:** These permissions do not behave like dangerous and normal permissions and are particularly sensitive. To access these permissions, a user must authorize, and the permissions must be declared in the manifest file. Examples include

permissions to access the device's camera or microphone while the device is in use by another app or to access the device's SMS or call log.

A structural approach of the Android permission model is visualized in Figure 4, where a permission check is performed every time that an application requests access to a resource. The flow checks the permission categories, such as normal, signature, dangerous, or special, and only grants access if the request adheres to the protocols established for each category. This flow ensures that apps only have access to the resources and data needed to perform their intended functions while protecting the user's security and privacy.

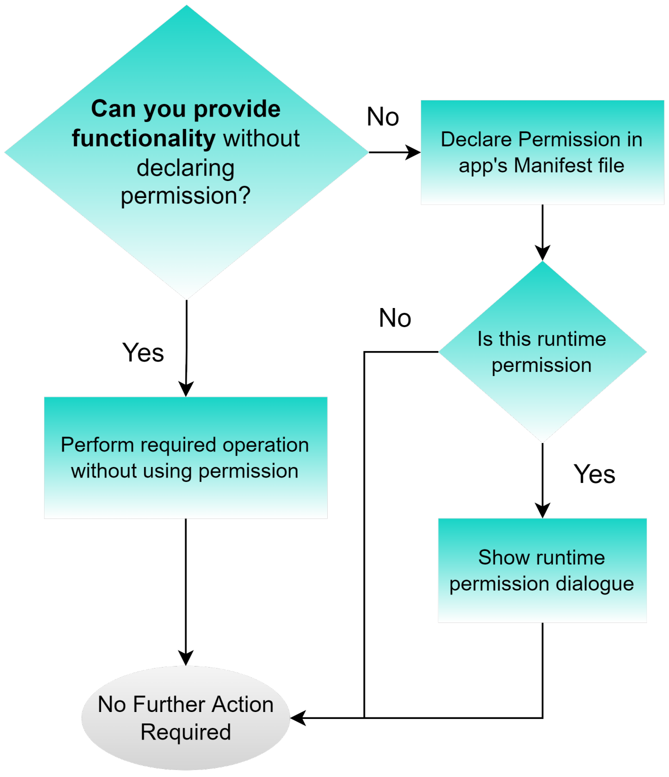

Can you provide functionality without declaring permission?
No
Declare Permission in app's Manifest file
Yes
No
Is this runtime permission
Perform required operation without using permission
Yes
Show runtime permission dialogue
No Further Action Required

Figure 4. High-level permission working model in Android.

3.6. *Android Application Package (APK)*

An APK is the file format used to distribute and install apps on the Android operating system [66]. It is a compressed file containing all the necessary files for an app to run on an Android device, including its code, resources, and Android manifest file. APK files are similar to other package file formats, such as *.exe* files on Windows or *.dmg* files on MacOS.

Users can install apps on their Android devices by downloading them from the Internet, transferring them via USB, or using an app store such as the Google Play Store. Once installed, the app can be launched and used like any other app. Developers can create APKs using the Android Studio development environment, which includes tools to create, test, and package apps in APKs. Developers can also sign their APKs with a digital signature to ensure the authenticity and integrity of the app [67,68].

An APK is made up of several components that are necessary for an app to function properly on an Android device. Details of these components are shown in Figure 5. The figure illustrates the file structure of APK, and details of the individual components are presented in the following paragraphs.

- **The app code:** This includes the Java or Kotlin source code that makes up the app's functionality. This code controls the app's behavior and handles user interactions.
- **Resources:** These are the files the application uses, such as images, layouts, and strings. These resources define the app's appearance and provide text for different languages.

- **The Android manifest:** This is a special XML file that contains important information about the app, such as the app's name, version, and permissions it requires. The manifest also defines the app's components, such as activities, services, and broadcast receivers.
- **External libraries:** If the app uses external libraries, those are also packaged inside the APK file. These libraries provide additional functionality and are typically open source projects integrated into the application.
- **Assets:** These are the files that the application uses but does not compile, such as fonts, audio, and video files. They are stored in the APK in their original format, and the app reads them at runtime.
- **Android runtime (ART):** This component runs the app on Android devices; it converts the app's bytecode into machine code so the device's processor can execute it.

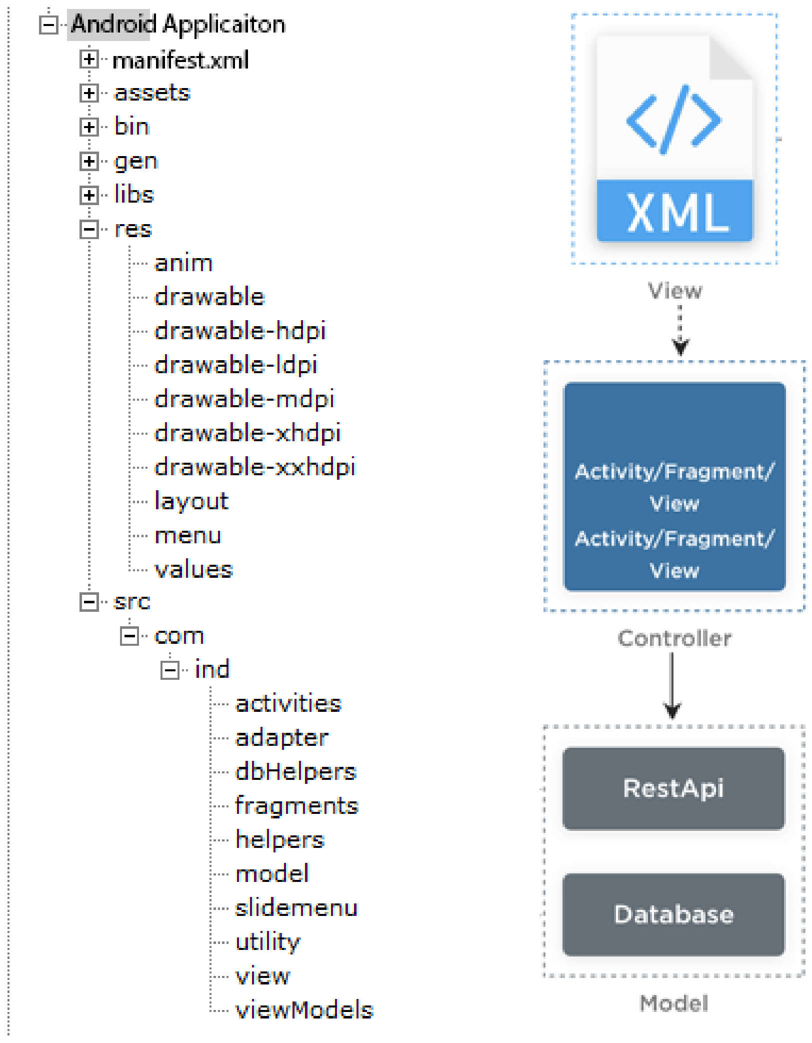


Figure 5. Illustration of an Android application package file structure.

3.7. Android Application Bundle (AAB)

An AAB is a new format introduced by Google for distributing Android apps. AAB is a new replacement for APK introduced on 6 August 2018, with Android 9 [69], but it is designed to be more efficient and flexible.

The main difference between an AAB and an APK is that an AAB contains all of the app's code and resources but does not include the compiled machine code for all devices. Instead, it includes the app's code and resources in a form that can generate machine code for specific devices during installation. This makes the app smaller and more efficient, as it only includes the machine code needed for the specific device on which it is being installed. Furthermore, AAB allows developers to build multiple APKs from a single Android project, which can be targeted at different devices, screen configurations, and languages, reducing the size of the final APK and improving the user experience.

3.8. Hardware/Software Platform and Processor Architectures

The combination of hardware devices and software components, such as device processors, storage units, LED displays, cameras, and batteries, manufactures Android devices. While software components may include Android architecture, SE-Linux, drivers, web browsers, games, and utility and productivity applications [70]. In Android, various types of processor architectures are used, the details of which are as follows [71]: (1) Application-specific integrated circuits (ASICs) are optimized for specific functions or applications that need high performance and low power consumption. They are hardcoded and not reprogrammable once manufactured. (2) Field programmable gate arrays (FPGAs) are programmable after manufacturing. They can be reprogrammed to integrate different configurations. These chips offer low manufacturing cost and relatively good performance [72]. Finally, (3) ARM/RISC-V are instruction set architecture (ISA) examples. These processors can be implemented as ASICs or FPGAs depending on the application's needs; they are optimized, have low power usage, and offer higher customization options [73].

4. Methods: Survey Methodology

This survey collects and analyzes articles from the past 15 years (2009–2023) focusing on smartphone security. To obtain the articles for this survey, we conducted a comprehensive literature search in five reputable databases: *ScienceDirect*, *IEEE Xplore*, *SpringerLink*, *Google Scholar*, and *Microsoft Academic*. We selected these databases due to their rich repository of research papers on various topics, which increases the visibility of research papers in smartphone security.

Our search keywords included *smartphone security*, *Android security*, *mobile vulnerabilities*, *cyber threats to mobile devices*, *sensor-based attacks*, *side-channel attacks*, *biometric attacks*, and *Google Play attacks*. We obtained 128,000 records from these keywords, further narrowed by excluding duplicates, becoming topic-specific, and removing poor writeups and non-journal/conference papers. We reviewed around 200 articles in detail, which were closely related to our survey topic and contributed vital knowledge to the field of smartphone security. These articles covered a wide range of topics, including the latest attacks and incidents in the field, and offered valuable information on the current state of smartphone security.

Among these 200 articles, 19 survey articles, at least one each year (2009–2023), are comparatively analyzed in the present survey to examine the strengths and weaknesses of existing research, current trends, and future directions. Details of these reviewed articles were added in ensuring paragraphs, and a quick overview is provided in Table 1. The table provides a detailed comparison of the related works in terms of the research objectives and contributions, and it provides a comprehensive overview of existing research and highlights the important contributions made by each study.

5. Android Vulnerabilities

A vulnerability refers to a weakness or flaw in the operating system or software that attackers can exploit to gain unauthorized access or control of a device or system. Vulnerabilities can exist in various forms, such as bugs in the code, outdated software, or misconfigured settings [74–76]. In recent years, Android has been reported with several vulnerabilities that may or may not be exploited by an attacker to gain access to sensitive information, such as personal data or financial information. Some vulnerabilities may also be exploited to install malware or other malicious software, compromising the device's security and privacy. Furthermore, vulnerabilities can launch attacks on other devices or systems, such as denial-of-service attacks or the spread of malware [33,76,77].

According to the common vulnerabilities and exposures (CVE) database, many vulnerabilities have been identified in the Android operating system since its beginning. Around 5000 vulnerabilities have been identified in the Android OS since 2009 [78–80]. Figure 6 provides details of several vulnerabilities identified in recent years. It can be seen that the number of vulnerabilities identified in the Android OS has been steadily increasing over the years [81].

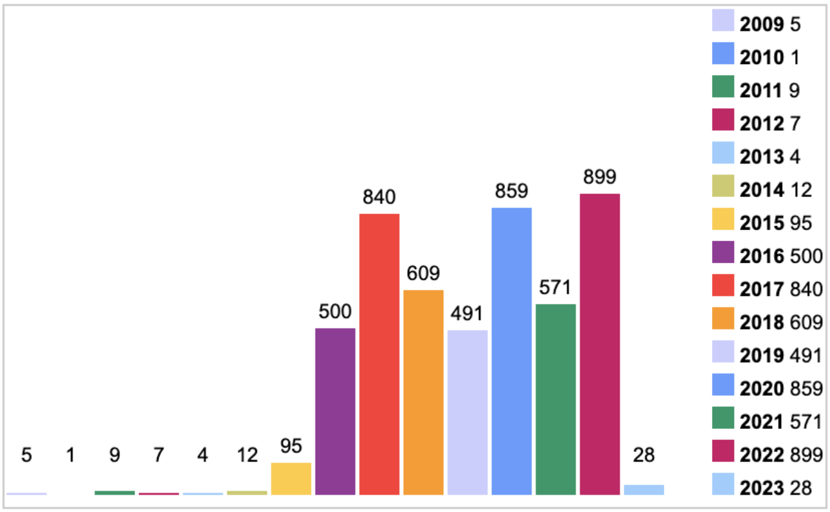


Figure 6. Graphical overview of vulnerabilities identified each year.

These numbers demonstrate the ongoing challenges the Android OS faces regarding security. Despite the efforts of developers and researchers to improve the security of the Android operating system, the number of vulnerabilities identified continues to increase. Furthermore, these identified vulnerabilities can be divided into various types, such as code execution, buffer overflow, and information gain. This categorization helps identify which types of vulnerabilities are the most popular and occurring. Figure 7 provides details of vulnerabilities by various types. The figure highlights that the most common vulnerabilities that lead to exploitation are code execution and buffer overflow. The details of the reported types of vulnerabilities are as follows:

1. **Denial of service (DoS):** This type of vulnerability occurs when an attacker floods a system with traffic or requests to overwhelm its resources and make it unavailable to legitimate users.
2. **Bypass something:** This vulnerability allows an attacker to bypass a security control, such as authentication or authorization, gain access to protected resources, or perform unauthorized actions.
3. **Execute code:** This vulnerability allows an attacker to execute an arbitrary code on a target system or device, which could lead to data theft, unauthorized access, or other malicious activities.
4. **Memory corruption:** This vulnerability involves exploiting bugs or flaws in a program's memory management system, such as buffer overflows or use-after-free errors, to gain unauthorized access to data or execute malicious code.
5. **Cross-site scripting (XSS):** This vulnerability allows an attacker to inject malicious code, such as JavaScript, into a web page or application, potentially leading to data theft or other malicious activities.
6. **Information disclosure:** This vulnerability allows attackers to access sensitive information without authorization, such as passwords, personal data, or system configuration information.

7. **Privilege escalation:** This vulnerability allows an attacker to gain higher access or privileges than authorized, potentially allowing them to perform malicious actions or access sensitive data.
8. **Buffer overflow:** This vulnerability occurs when an attacker inputs more data into a program's memory buffer than it can handle, potentially leading to the execution of arbitrary code or a system crash.
9. **SQL injection:** This vulnerability allows an attacker to inject malicious SQL code into a web application or database, potentially leading to unauthorized access or data modification.
10. **Directory traversal:** This vulnerability involves exploiting a web application's file path validation flaw to gain unauthorized access to files or directories outside the application's intended scope.

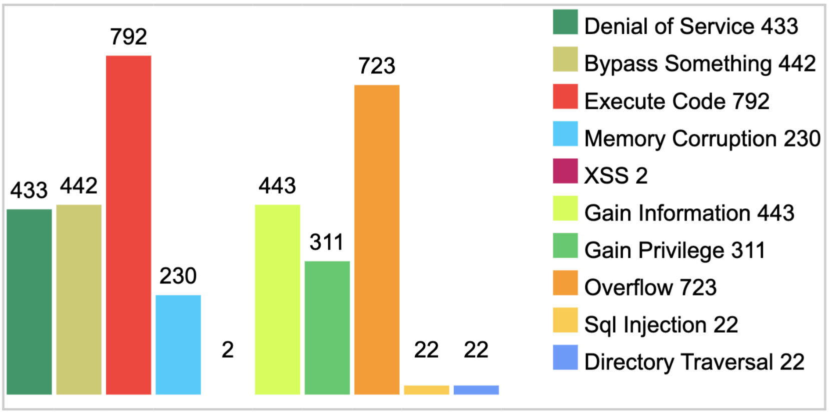


Figure 7. Classification of Android vulnerabilities by vulnerability types.

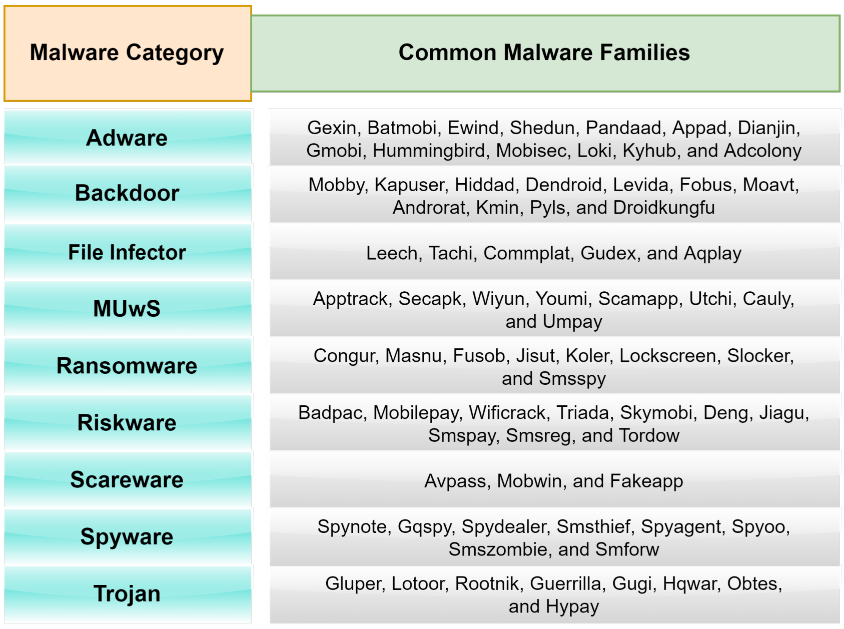
This information can be used to prioritize efforts to address vulnerabilities and identify patterns in the types of vulnerabilities most commonly targeted. In addition, it can provide information on how the industry is evolving and allow pro-active measures to be taken to address the vulnerabilities that are most likely to be targeted in the future.

6. Cyber Threats

Cyber threats are malicious attacks or attempts to exploit vulnerabilities in mobile devices. With regard to cyber threats, here we refer to all software-based threats that exploit an existing vulnerability in OS, as well as any attempt that is made to trick users into giving away personal information or downloading malware [15,82]. On a broader level, the identified threats are divided into three major categories: contemporary threats, rare cyber threats, and APTs. The details and categorization of each have been added in ensuring Sections [83–85].

6.1. Contemporary Threats

Contemporary cyber threats are the current and most prevalent security risks that affect smartphones and tablets. These threats can take many forms and potentially compromise personal and sensitive information. Multiple variants of these contemporary threats are addressed in Figure 8. The figure provides a high-level classification of contemporary malware families and variants used in recent attacks. For instance, the Adware category contains multiple families: gain, hummingbird, kyhub, and adcolony. The details of all these threats are provided in subsequent Sections [32,86].



Malware Category	Common Malware Families
Adware	Gexin, Batmobi, Ewind, Shedun, Pandaad, Appad, Dianjin, Gmobi, Hummingbird, Mobisec, Loki, Kyhub, and Adcolony
Backdoor	Mobby, Kapuser, Hiddad, Dendroid, Levida, Fobus, Moavt, Androrat, Kmin, Pyls, and Droidkungfu
File Infector	Leech, Tachi, Commplat, Gudex, and Aqplay
MUwS	Apptrack, Secapk, Wiyun, Youmi, Scamapp, Utchi, Cauly, and Umpay
Ransomware	Congur, Masnu, Fusob, Jisut, Koler, Lockscreen, Slocker, and Smsspy
Riskware	Badpac, Mobilepay, Wificrack, Triada, Skymobi, Deng, Jiagu, Smspay, Smsreg, and Tordow
Scareware	Avpass, Mobwin, and Fakeapp
Spyware	Spynote, Gqspy, Spydealer, Smsthief, Spyagent, Spyoo, Smszombie, and Smforw
Trojan	Gluper, Lotoor, Rootnik, Guerrilla, Gugi, Hqwar, Obtes, and Hypay

Figure 8. Classification of contemporary threats targeting mobile devices.

6.1.1. Adware

Adware is malware that displays unwanted advertisements on the device, often pop-ups. These ads can be difficult to remove and may continue to display even after removing the adware. Adware can spread through infected apps or websites [87].

6.1.2. Backdoor

A backdoor is a code that allows unauthorized access to a device. This can execute malicious commands on the device, such as stealing sensitive information or installing malware [6].

6.1.3. File Infector

File infectors are malware that attach themselves to files, such as APKs, to access an application's data [88,89]. This can allow malware to conduct harmful actions, such as stealing sensitive information or encrypting files.

6.1.4. Mobile Unwanted Software (MUwS)

MUwS refers to programs that have a negative impact on the software ecosystem. They can imitate other applications or capture user information without permission [90]. Examples include applications that display unwanted advertisements or track user activity.

6.1.5. Ransomware

Ransomware is a type of malware that encrypts user data and demands payment in exchange for access to encrypted files. Ransomware can spread through infected websites or spam emails [91].

6.1.6. Riskware

Riskware is a type of legitimate software that presents potential risks to device security. Bad actors can use riskware to steal information or redirect users to malicious websites [92].

6.1.7. Scareware

Scareware is a type of malware that uses fear to manipulate users into downloading or purchasing harmful programs. Examples include fake security software that claims to protect the device [93].

6.1.8. Spyware

Spyware is a type of malware that tracks user activity and steals personal information. It can run in the background and may be difficult to detect [94].

6.1.9. Trojan

A trojan is a type of malware that disguises itself as a legitimate program, such as a game. However, it can perform harmful actions, such as tracking user activity and stealing personal information. Trojans can be spread through infected apps or websites [95].

6.2. Advanced Persistent Threats (APTs)

APTs are a specific type of targeted cyberattack that aims to infiltrate a specific target and remain undetected for an extended period of time [96]. These attacks are often carried out by well-funded and highly skilled attackers, such as nation-state actors or organized cybercriminal groups. The objective of APTs is not just to infect as many devices as possible but to gain access to a specific target's device to carry out cyber espionage, data theft, or sabotage [97]. APTs on Android are designed to be stealthy and use sophisticated tactics such as social engineering, zero-day exploits, and spear phishing to gain access to a target's device [98]. Here are some of the APTs that were detected in recent years.

6.2.1. Pegasus

Pegasus is a highly sophisticated spyware that infects Android and iOS devices through various methods, including spear phishing and zero-click attacks. Developed by the Israeli cyber arms firm NSO Group, it can collect a wide range of data, including text messages, call logs, and location data [99]. Pegasus exploits several vulnerabilities in the operating system and various apps, such as WhatsApp, commonly used by targeted individuals.

6.2.2. Gooligan

Gooligan is an APT typically spread through malicious apps and can take control of infected devices [100]. Once installed, the Gooligan can steal authentication tokens and gain access to Google accounts, allowing the attackers to install additional malware and steal sensitive data.

6.2.3. Dark Caracal

Dark Caracal is an APT group known to target Android devices with a malicious app called "Secure Android". Once installed, Secure Android can collect a wide range of data from the infected device, including call logs, text messages, and audio recordings [101].

6.2.4. Hornbill

Hornbill is an Android APT malware capable of stealing sensitive information from infected devices, including text messages, call logs, and contact lists [9]. It is mainly spread through phishing messages that contain links to malicious websites. Once malware is installed on a device, it can communicate with a remote server controlled by the attacker, allowing them to collect data.

6.2.5. SunBird

SunBird is a remote access trojan (RAT) designed to steal sensitive information from infected Android devices, including SMS messages, call logs, and contact lists. It is spread primarily through phishing emails and SMS messages that contain malicious links [102]. Once a user clicks on the link, the malware is downloaded to their device, where it can remain hidden while stealing data and communicating with a remote server controlled by the attacker.

6.2.6. Skygofree

Skygofree is an APT capable of recording audio and video, taking screenshots, and stealing sensitive data from infected devices [103]. Skygofree is typically spread through fake websites that prompt users to download and install the malware on their devices. Once installed, malware can gain root access to the device, making it difficult to detect and remove.

6.2.7. Triout

Triout is an APT designed to steal data from infected devices, including call logs, text messages, and location data [104]. Triout is typically spread through malicious apps disguised as legitimate applications. Once installed, the malware can take control of the device and communicate with command-and-control servers to download additional malware or steal sensitive data.

6.2.8. Mosaic Regressor

Mosaic regressor is an APT that targets Android devices. It is typically spread through phishing messages that contain links to malicious websites [105]. Once installed on a device, malware can collect a wide range of data, including call logs, text messages, and location data. The mosaic regressor is also capable of taking screenshots and recording audio.

7. Threats through Google Play Store

The Google Play Store is the official app store for Android devices, where users can download and install apps, games, and other types of software. The apps in the store are developed by third-party developers and are reviewed by Google before being made available to the public [106]. Google has implemented a Play Protect tool to scan apps for malware and other malicious content before publishing them in the store. This includes the static and dynamic analysis of the apps and checking for any known malicious behavior. However, despite these measures, attackers find their way into the store [18,51,107]. There are several means by which attackers can upload and propagate malware through the Google Play Store, the details of which are as follows.

1. **Dynamic code loading:** Attackers can use dynamic code loading techniques, such as Java reflection or Android DexClassLoader, to load and execute code at runtime [108]. This can be used to download and execute additional code or malicious payloads after the app has been installed, which can be hidden in the app's legitimate code or downloaded from a remote server. This can allow attackers to evade detection by security scanners, as the malicious code may not be present in the initial app release.
2. **Incremental malicious updates attack (IMUTA):** Attackers can use incremental updates to gradually add malicious code to an application over time [18]. This can be performed using a dropper app that initially appears legitimate but later downloads and installs additional components or payloads. Attackers can also use code obfuscation and dynamic code loading techniques to add malicious code in a way that is difficult to detect, such as updates. This can allow attackers to evade detection by security scanners and prolong the lifespan of the malware.
3. **Code obfuscation:** Obfuscation consists of modifying an app's source code to make it more difficult to understand or analyze [109]. Attackers can use obfuscation techniques, such as renaming variables and classes, adding junk code, or using encryption to hide the functionality and purpose of the malicious code. This can make it more difficult for security researchers to detect and analyze malware [110].
4. **Repackaging:** Attackers can use repackaging techniques to take a legitimate app and add malicious code. This can be performed by decompiling the application, adding malicious code, and then recompiling and resigning the application [51]. The repackaged app can then be uploaded to the Google Play Store, and users may be manipulated into downloading it, as it may appear legitimate and have good reviews.

5. **Social engineering:** Social engineering manipulates people into performing certain actions or divulge sensitive information. Some cybercriminals use social engineering to trick users into downloading and installing applications that appear to be legitimate but are malware. These applications can look like popular apps or games but contain malware that can steal personal information or perform other malicious actions [51,59].

The Google Play Store hosts numerous malware families, which have been identified, reported, and published. Google has detected and blocked many Android malware families, but many recent incidents still evidence security loopholes and vulnerabilities in security mechanisms implemented by the Google Play Store.

Table 2 lists 50 malware families propagated using the Google Play Store [23,111,112]. These malware families are not limited to a single application or account, and their code can be found to be associated with hundreds of applications which users can unwittingly download. As such, it is crucial that users and developers exercise caution and diligence when using and creating apps, respectively, and that the Google Play Store continues to implement robust security measures to detect and prevent the spread of malware.

Table 2. List of Android malware families detected on the Google Play Store [23,113,114].

No.	Family Name	Number of Applications	Overall Percentage (%)
1	Airpush	1574	23.8
2	Plankton	722	10.9
3	Adwo	593	9.0
4	Kuguo	492	7.4
5	Leadbolt	298	4.5
6	Dowgin	261	3.9
7	Fakeinst	214	3.2
8	Gingermaster	194	2.9
9	Wapsx	192	2.9
10	Waps	187	2.8
11	Youmi	110	1.7
12	Smsreg	95	1.4
13	Utchi	84	1.3
14	Lotoor	80	1.2
15	Smssend	71	1.1
16	.Admogo	65	1.0
17	Agent	62	0.9
18	Smsspy	59	0.9
19	Boqx	57	0.9
20	Wooboo	54	0.8
21	Mseg	36	0.5
22	Droidkungfu	28	0.4
23	Mobwin	27	0.4
24	Immodiads	26	0.4
25	Nandrobox	25	0.4

Table 2. *Cont.*

No.	Family Name	Number of Applications	Overall Percentage (%)
26	Smspay	25	0.4
27	Mulad	23	0.3
28	Opfake	23	0.3
29	Ginmaster	22	0.3
30	Andup	20	0.3
31	Adload	20	0.3
32	F47v1119	20	0.3
33	Umeng	19	0.3
34	Domonn	19	0.3
35	Kyview	18	0.3
36	Viser	18	0.3
37	Mobclick	16	0.2
38	EEE	16	0.2
39	Ksapp	15	0.2
40	Revmob	14	0.2
41	Droidtrooter	14	0.2
42	Smskey	12	0.2
43	Umpay	10	0.2
44	Bankun	9	0.1
45	Goldentouch	8	0.1
46	Adswo	7	0.1
47	Chuli	6	0.1
48	Hacktool	5	0.1
49	Telman	4	0.1
50	Ddlight	3	0.0

These malware families can reside inside different applications on the Google Play Store in several ways. Once malware is downloaded and installed on a user's device, it can perform various malicious actions. This can include stealing personal information, displaying unwanted ads, or even taking control of the device. These malware attacks can be harder to detect by Google Play Protect and other mobile antiviruses since they are embedded inside the apps and may not have any visible signs of malicious behavior. Malware can also be designed to run only after a certain time, making it harder for antiviruses to detect.

Interestingly, malware can be detected by analyzing the permissions they request when installed on an Android device. Analyzing the permissions requested by a particular application makes it possible to determine whether it is potentially malicious. For example, if an app requests permissions unrelated to its intended functionality, such as the ability to access the user's contacts or call logs, it may be a sign that the app is malicious. Additionally, if an app requests permissions that give it access to sensitive information, such as location data, it may be a sign of malware. Table 3 provides a useful resource for malware detection by listing the most frequent permissions that malware requests when installed. It can be a useful resource for detecting malware by listing the most frequent permissions that malware requests when installed. The table has four columns:

Table 3. List of the most common permissions requested by Android malware [23,113,114].

Family	%	Group	Severity
INTERNET	98.8	Network	Dangerous
ACCESS_NETWORK_STATE	94.4	Network	Normal
READ_PHONE_STATE	82.7	Phone_Call	Dangerous
WRITE_EXTERNAL_STORAGE	70.4	Storage	Dangerous
ACCESS_COARSE_LOCATION	56.2	Location	Dangerous
ACCESS_WIFI_STATE	54.9	Network	Normal
ACCESS_FINE_LOCATION	53.7	Location	Dangerous
GET_ACCOUNTS	29.0	Accounts	Normal
C2D_MESSAGE	24.7	None	Signature
RECEIVE_BOOT_COMPLETED	24.7	App_Info	Normal
SYSTEM_ALERT_WINDOW	17.3	Display	Dangerous
READ_EXTERNAL_STORAGE	14.8	Storage	Normal
CALL_PHONE	14.2	Phone_Calls	Dangerous
CAMERA	13.0	Camera	Dangerous
RECORD_AUDIO	12.3	Microphone	Dangerous
READ_HISTORY_BOOKMARKS	11.7	Bookmarks	Dangerous
SEND_SMS	11.7	Messages	Dangerous
READ_LOGS	6.8	Tools	Signature
READ_CONTACTS	4.9	Social_Info	Dangerous
UNINSTALL_SHORTCUT	3.7	System_Tools	Dangerous
RECEIVE_SMS	2.5	Messages	Dangerous
WRITE_CONTACTS	2.5	Social_Info	Dangerous
MANAGE_ACCOUNTS	2.5	Accounts	Dangerous
READ_SMS	2.5	Messages	Dangerous
WRITE_SMS	1.9	Messages	Dangerous
USE_CREDENTIALS	1.9	Accounts	Dangerous
PROCESS_OUTGOING_CALLS	1.9	Phone_Calls	Dangerous
MODIFY_PHONE_STATE	1.2	Phone_Calls	Signature
ACCESS_SUPERUSER	0.6	None	Dangerous
DOWNLOAD_WITHOUT_NOTI	0.6	None	Dangerous
WRITE_CALL_LOG	0.6	Social_Info	Dangerous
ACCESS_DOWNLOAD_MANAGER	0.6	None	Normal
DELETE_PACKAGES	0.6	None	Signature
RECORD_VIDEO	0.6	None	Dangerous
READ_CALL_LOG	0.6	Social_Info	Dangerous
DELETE_CACHE_FILES	0.6	None	Signature
CHECK_LICENSE	0.6	None	Normal

- **Permission name:** This column highlights the permission name used in mobile devices. It lists the different permissions that malware may request when installed on an Android device, such as Internet access, camera access, calendar access, call logs, contacts, messages, and location.
- **Percentage of use:** This column highlights the percentage of the use of a specific permission by various malware attacks. It shows the frequency with which particular permissions are requested by malware, which can be used to identify potentially malicious apps.
- **Permission group:** This column groups the permissions into categories. This can help identify patterns in the permissions requested by malware and provide insight into the malware's functionality.
- **Classification:** This column classifies the permissions based on the Android permission model. This can help one understand the level of access that malware has to the device and the potential severity of the malware.

By analyzing Tables 2 and 3, security experts and researchers can identify potentially malicious apps by analyzing the permissions they request. It is also important to note that not all apps that request these permissions are malicious, but it can be a red flag that needs to be investigated further.

8. Sensor-Based Attacks

Sensor-based attacks are a prevalent cyberattack targeting sensors embedded in Android devices. These attacks are designed to exploit the device's sensors, such as GPS, camera, microphone, accelerometer, and others, to extract sensitive information or gain unauthorized access to the device. The attackers employ various techniques to achieve their objectives, such as tracking the device's location, recording audio or video, and extracting sensitive information. These attacks can have severe consequences, such as identity theft, financial loss, and unauthorized access to confidential data. Below, we present several Android Sensor-based Attacks.

8.1. Global Positioning System (GPS) Attacks

GPS is a global navigation satellite system that provides location and time information for various applications, including navigation, surveying, tracking, and mapping. GPS is also commonly used on mobile devices [115]. GPS attacks on Android devices refer to various methods and techniques attackers use to exploit vulnerabilities in the GPS sensor or the location-based services that use it [115]. These attacks can take many forms, such as tracking the location of a device without the user's knowledge or consent, spoofing GPS coordinates to mislead the user or a service, or using GPS data to perform other malicious activities [116,117]. These attacks can have severe consequences, such as privacy violations, location-based scams, or even physical harm in some cases [118]. Moreover, malicious applications with GPS information rights can log individuals with precise location coordinates and track their movements [119].

8.2. Near-Field Communication (NFC) Attacks

NFC is a technology for wireless communication between devices close to each other, which are typically no more than a few centimeters apart. It operates at a frequency of 13.56 MHz and uses magnetic field induction to communicate between devices [120]. NFC is based on radio frequency identification (RFID) technology and is used for various applications, including contactless payments, data transfer, and access control. An NFC attack exploits vulnerabilities in NFC-enabled devices or the NFC protocols themselves. These attacks can take various forms, such as eavesdropping on data transmitted over NFC, modifying or injecting data into NFC communications, or impersonating a legitimate NFC device [121]. Some examples of NFC attacks include skimming (stealing payment card data from contactless cards), relay attacks (relaying data from an NFC-enabled device to a criminal's device), and cloning (copying the data from a legitimate NFC device to a malicious device) [122]. These attacks can be carried out using specialized hardware or software tools and compromise personal and financial information security.

8.3. Battery-Draining Attacks

Battery-draining attacks in Android smartphones refer to a method of draining a device's battery power by running malicious software or applications that consume a significant amount of energy. This type of attack can be used to disable a device or prevent it from being used during a period of time. These attacks can also prevent the device from being used for legitimate activities such as making phone calls, sending text messages, or accessing the Internet [123]. In such attacks, an attacker utilizes maximum resources to drain the device's power [124,125]. Some of these attacks include crypto mining, cryptojacking, the frequent disconnect of the Wi-Fi network, spam propagation, background services, and unnecessary system resource utilization [26,126–129].

8.4. Wi-Fi Attacks

Wireless fidelity (Wi-Fi) technology allows electronic devices to connect to a wireless network using radio waves. It is a popular Internet connection method, as it is widely supported on many devices, including smartphones and laptops [130]. On the other hand, Wi-Fi attacks on Android mobile devices refer to a wide range of malicious activities that target wireless network connections on these devices. Wi-Fi attacks are a growing concern for Android mobile devices, as they allow attackers to steal sensitive information and launch further attacks by manipulating wireless networks. A Belgian mobile scientist M. Vanhoef demonstrated an attack and named it "Frag Attacks" [131]. A regression attack allows an attacker to steal information through a secure network and launch a DoS attack. Moreover, using DNS hijacking, an attacker can redirect user traffic to spoofed targeted websites such as fake login and payment pages [132]. Some more example attacks can include the following:

- **Jamming signals:** An attacker can use a device to transmit radio waves at the same frequency as the wireless network, causing interference and preventing legitimate devices from connecting. This type of attack is known as a jamming attack, and it can be executed using a variety of devices, including portable jammers, software-defined radios, and even smartphones.
- **Evil twin:** This is an attack in which a malicious wireless network is set up to mimic a legitimate network to trick users into connecting. Once connected, the attacker can access sensitive information or infect the device with malware. This type of attack is also known as an "evil twin" or "rogue access point" attack.
- **Man-in-the-middle (MitM) attack:** This attack is where the attacker intercepts communications between the device and the wireless network, allowing them to access sensitive information or launch further attacks. MitM attacks can be executed using various techniques, including ARP spoofing, DNS spoofing, and SSL stripping.
- **Rogue access points:** This type of attack is similar to an "evil twin" attack, where the attacker sets up a rogue wireless access point and tricks users into connecting to it. Once connected, the attacker can access sensitive information or launch further attacks.
- **Honeypot attack:** An attacker can set up a free Wi-Fi connection in a public place or open areas, such as a street, park, or coffee shop, to interfere with the traffic of the connected device. This type of attack is known as a "honeypot" attack, designed to lure unsuspecting victims into a trap.

8.5. Gyroscope Attacks

Mobile devices use a gyroscope sensor to measure angular velocity, motion, orientation, tilt, and device rotation to smooth the user experience and adjust the display accordingly. Gyroscope attacks in Android refer to a security vulnerability that allows an attacker to gain access to sensitive information, such as the rotation and orientation of the device, by exploiting the gyroscope sensor. This can potentially be used to track the device's location and gain insight into the user's activities and movements [133]. In addition, gyroscopes can be used to launch keyboard vibration attacks, which can be used to exfiltrate data from air-gapped systems through smartphones. As reported by Guri et al. at an international conference on privacy, security, and trust (PST) [134], they have demonstrated a proof of concept in which they show how an attacker can use speakers to launch real-time communication with the gyroscope and exfiltrate sensitive data from an air-gapped system. Furthermore, gyroscopes can be used with speech recognition software to enable eavesdropping when microphones are unavailable [135].

8.6. Biometrics Attacks

Biometrics refers to using physiological or behavioral characteristics to identify or authenticate an individual. Examples of biometric characteristics include fingerprints, facial features, iris patterns, and voice [136]. Biometric authentication mechanisms mainly unlock the device or access secure apps and data. However, biometric attacks in Android devices refer to methods used by attackers to bypass or defeat the biometric authentication measures put in place to protect devices and data. These attacks can take various forms, including:

- **Deepfake attack:** A deepfake attack is a biometric attack that uses AI-generated images, videos, or audio to impersonate a real person and fool the biometric sensor into thinking it is the legitimate user. These attacks can bypass facial, voice, or even iris recognition systems.
- **Presentation attacks:** Presentation attacks involve a real biometric characteristic, but the individual is not the legitimate user. For example, using someone else's fingerprint or face to access the device. It involves recording a legitimate user's biometric characteristics and then using them to gain access to the device.
- **Fingerprint hijacking attack:** A fingerprint hijacking attack is a type of biometric attack that involves the use of physical replicas of fingerprints, such as 3D-printed fingerprints or lifted fingerprints, to fool the fingerprint sensor into thinking it is the legitimate user. This can be performed by tricking the user with specially designed sticky paper, glass, or third-party hardware coatings [137]. Afterward, this unique finger impression can deceive the device finger scanner into performing an unauthorized login [138].
- **Voice-based attacks:** Voice-based attacks are a type of biometric attack that involve using AI-generated speech or voice recordings to impersonate a real person and fool the voice recognition system into thinking it is the legitimate user. Nowadays, individual assistants such as Alexa, Siri, Cortana, and Google Home are broadly used. Scientists have performed an attack demonstration to prove that smartphones can be attacked using hidden voice commands. These voice commands are not understandable by humans [61,139]. This stealthy attack can be launched on a mobile device without user awareness. Furthermore, this attack can be launched to make unauthorized calls, share information on social media, send targeted messages, and many more [140].
- **Facial recognition attacks:** Facial recognition attacks are a type of biometric attack that involves the use of physical replicas of faces, such as 3D-printed faces or photographs, to fool the facial recognition system into thinking it is the legitimate user. Most cutting-edge facial biometric frameworks are ineffective against basic attacks and lack attack detection measures [141]. The facial recognition module is misleading by showing the camera a photograph, video, or 3D cover of the individual [142].
- **Iris authentication attacks:** Iris recognition attacks are a type of biometric attack that involves the use of replicas of iris patterns, such as 3D-printed irises or photographs of irises, to fool the iris recognition system into thinking that it is the legitimate user. Iris recognition sensors are surprisingly powerless against cyberattacks. They can be bypassed using bogus blueprints and similar matching. One of the least demanding ways to trick an iris scanner is by showing a printed picture of a member's iris, using

an electronic screen such as a cell phone, or, in any event, using a contact lens to trick the framework [143,144].

9. Side-Channel Attacks

Side-channel attack refers to a type of attack that is used to extract sensitive information from a mobile device from a source other than the device's primary input or output mechanisms. This may include analyzing mobile physical characteristics such as power consumption, electromagnetic radiation, or sound emitted by the device while performing certain operations. Side-channel attacks can extract sensitive information from the device, such as encryption keys, passcodes, and personal information. These attacks can be launched by analyzing generated patterns such as vibration, movement, and screen blinks. Several types of side-channel attacks can be launched against Android smartphones. These attacks include:

9.1. Smudge Attacks on Touch Screens

Smudge attacks on touch screens in Android refer to bypassing the lock screen on an Android device by analyzing the fingerprints left behind on the touchscreen by the device's owner [145]. By analyzing these fingerprints, an attacker can recreate the swipe pattern or PIN used to unlock the device, allowing them to access the device's data. This type of attack is a form of physical access attack, as it requires the attacker to have physical access to the device to perform the attack. This attack is not only limited to smartphone touchscreens. Still, it applies to all devices that use secret passwords, patterns, pin line automated teller machines (ATMs), Internet of Things (IoT), and other touchscreen interfaces. The phenomenon is also called touchscreen eavesdropping [146–148].

9.2. Motion-Based Keystrokes Attacks

Motion-based keystroke attacks on Android smartphones refer to extracting sensitive information, such as passwords or PINs, by analyzing motion sensors on a device, such as an accelerometer or gyroscope. By analyzing the motion data of the device while the user is typing, an attacker can determine the keys being pressed and thus infer that the password or PIN has been entered. Typing on a flexible mobile phone keyboard produces various vibrations that can be mishandled to distinguish the key being pressed [135,149,150]. An attacker demonstrated an attack named AlphaLogger, an Android-based application that can induce letter keys, typed on a soft keyboard [16]. AlphaLogger runs in the background and collects information at 10 Hz/s from cell phone sensors such as an accelerometer, gyroscope, and magnetometer to accurately induce keystrokes typed on the keyboard of all other applications running in the foreground. The results show that keystrokes can be predicted with 90% accuracy [151].

9.3. Password Inference Using Accelerometer

The accelerometer is a sensor that measures the device's acceleration, stabilization, orientation, and detection of shaking. Password inference using an Android smartphone accelerometer refers to the extraction of sensitive information, such as passwords or PINs, by analyzing the motion data of a device. By analyzing the motion data of the device while the user is typing, an attacker can determine the keys being pressed and thus infer that the password or PIN has been entered. This sensor has a powerful auxiliary channel that can extract the entire text sequence entered on a touchscreen cell phone keyboard [152]. Accelerometer estimation can be used to separate six-character passwords. Using this sensor allows an attacker to monitor sensitive activities, such as account logins and other device credentials [153,154].

9.4. Juice Jacking Attacks

Juice jacking attacks typically involve malicious charging cables, USB ports, and wall adapters designed to look like legitimate charging stations [155]. These malicious devices are modified to charge the device and install malware or steal data. The malware can be used to gain control of the device, steal personal information, and even install ransomware. Data stolen by these malicious devices can include passwords, credit card numbers, and other sensitive information. These malicious charging devices can also bypass security measures such as two-factor authentication and encryption. Malicious ports are usually disguised as public charging stations but can also be found in hotel rooms, airports, and other public spaces [156,157].

9.5. Android Fault Attacks

Fault attacks are physical attacks that manipulate a device's hardware to induce errors or faults in its regular operation. These faults can be exploited to bypass security checks, leak sensitive information, or execute a malicious code [158]. Some of the common techniques for fault injection on Android devices are as follows:

1. **Voltage fault injection:** This involves manipulating the power supply of the device to cause glitches in the CPU or memory. For example, VoltJockey3 [159] is an attack that uses voltage scaling to inject faults in the ARM TrustZone and compromise the secure world execution.
2. **Electromagnetic fault injection:** This involves applying electromagnetic pulses to the device to induce transient faults in the circuits. For example, EM-Fuzz [160] is an attack that uses electromagnetic fault injection to fuzz Android kernel drivers and find vulnerabilities.

To prevent fault attacks on Android devices, hardware-based countermeasures can be used, which involve adding physical protection mechanisms to the device. For example, Google's Titan M security chip [161] has a built-in tamper detection and protection features that can resist physical attacks. Software-based countermeasures can also be used, which involve adding code-level protection mechanisms to the device. For example, Android's verified boot feature [50] can verify the integrity and authenticity of the boot chain and prevent loading corrupted or malicious code. To detect fault attacks, methods such as monitoring system behavior and analyzing system logs can be used.

9.6. Power Analysis Attacks

Power analysis attacks are side-channel attacks that exploit the power consumption of a device to infer sensitive information, such as encryption keys, passwords, or user activities [162]. These attacks can be classified into two major categories as (1) simple power analysis (SPA), which involves observing the power consumption of a device and identifying patterns or features that reveal information; and (2) differential power analysis (DPA), which involves collecting the multiple power traces of a device and applying statistical analysis to extract information.

To prevent power analysis attacks on Android smartphones, protection mechanisms must be used to safeguard the device. For example, Google's Titan M security chip [161] has built-in noise generation and filtering features that can resist power analysis attacks. Software-based countermeasures can also be used, including adding code-level protection mechanisms to the device. For example, Android's Keystore system [163] can use cryptographic techniques to protect encryption keys from power analysis attacks.

9.7. Fault Attacks on Cryptographic Algorithms

Android fault attacks can target different cryptographic algorithms that are used in Android devices for security and privacy purposes. These algorithms include block ciphers, hash functions, and stream ciphers. Some examples of these algorithms are the Pomaranch cipher, Grostl hash, Midori cipher, RECTANGLE cipher, and Welch–Gong (WG) [164]. Fault attacks can exploit the vulnerabilities of these algorithms to recover their secret keys or plain texts or to bypass their security properties. For example, fault attacks can inject faults into the architectures of the Pomaranch cipher, Grostl hash, Midori cipher, and RECTANGLE cipher to induce errors in their internal states or outputs [165].

These errors can then be used to perform differential fault analysis or algebraic fault analysis to recover the secret keys. Fault attacks can also inject faults into the error detection mechanisms of lightweight stream ciphers such as Welch–Gong (WG) to disable their protection and allow the attacker to reuse the same keystream for different messages.

10. Intrinsic Cyberattacks

Intrinsic cyberattacks are malicious attacks that are designed to exploit the vulnerabilities that exist within the hardware or software of a device. These attacks can be used to access sensitive information, disrupt operations, and damage the device. In the context of Android smartphones, intrinsic cyberattacks take advantage of security vulnerabilities found in the phone's operating system, apps, and hardware components. Examples of intrinsic cyberattacks on Android devices are outlined below.

10.1. Application Collusion Attack

An application collusion attack is carried out with the help of two or more applications. In this attack, an application uses the resources allocated by another application [166]. This is performed by splitting a malicious functionality within multiple applications and exfiltrates data by exploiting the inter-app communication feature [167]. First, the attacker splits malicious code among applications with the same application and process ID. Afterward, this attack is performed using the mobile OS' intra-apps communication feature. This attack is particularly effective because the user is typically unaware that multiple applications are being used to carry out the attack and may be more likely to trust and install the applications.

10.2. Inter-App Communication Attack

This attack can be classified as the subcategory of an application collusion attack, where multiple malicious applications work together to exploit system vulnerabilities. Android provides a feature called broadcast receivers, which allows applications to interact with each other [168–170]. However, this feature can also be exploited by malware to carry out various types of attacks, such as broadcast theft, activity hijacking, intent spoofing, service hijacking, and privilege escalation [171,172].

10.3. StrandHogg Attack

StrandHogg is an Android vulnerability (critical severity: CVE-2020-0096) through which a malicious application is installed on the operating system [173]. Afterwards, it can exploit and access system resources such as SMS, photos, login credentials, GPS coordinates, camera, and microphone. Furthermore, the installed application can make, record, and delete phone conversations [174,175]. It exploits Android multitasking features and leaves behind traceable markers, enabling simultaneous attacks that are difficult to detect. It works in the following way; first, a StrandHogg application is installed. It is launched when a legitimate application launch icon is clicked and overlayed. The user thinks that the original application is installed, but instead, the StrandHogg fake activity screen is displayed. Subsequently, the collected data are exfiltrated to the destination server [176].

10.4. Keystore Forgery Attack

Android keystore is a cryptographic key storage file container which is difficult to extract and modify from the application. It is designed to protect the security of the application [177,178]. Thus, a keystore forgery attack is performed to modify keystore data. In this attack, an attacker takes advantage of the length of the symmetric key and tries to modify it according to the requirements. This phenomenon is called breaking into the keystore [179]. The motivation behind a forgery attack is that a modified or malicious application can be signed with the same keystore file to appear legitimate to the user, the application distribution platform, and the mobile device [179]. For example, the attacker downloads an authentic game and repackages it with malware, and uploads the mobile application to an outsider application store, from where the end client downloads the malevolent game application, trusting it to be certified. Thus, malware gathers and sends client confidential data such as call logs, photographs, recordings, and documents to attackers without the client's information [178,180,181].

10.5. Application-Level Sandboxing Attack

Android employs a security mechanism known as "sandboxing", which effectively isolates an application's resources and prohibits it from interfering with or accessing the resources of other applications. Sandboxing works by assigning each application its own unique "sandbox" or isolated environment in which it can operate [182]. This approach is designed to restrict an application's access to the underlying system and other applications. Despite the robustness of sandboxing, malware applications can still attempt to exploit it by requesting unnecessary permissions, exploiting known vulnerabilities in the operating system, or using other methods to circumvent the restrictions imposed by the sandbox. For example, a malware application may request access to sensitive data or resources, such as contacts or cameras, and then utilize that access to extract the data or perform other malicious actions [183]. Additionally, certain malware applications may also leverage techniques such as code injection or reflection to evade the sandbox's limitations and access the resources of other applications.

10.6. Android Application Layer Attack

The application layer is the hardest to protect in mobile devices. This is because applications frequently depend on complex input from clients. These inputs are difficult to characterize with intercepts, soft checks, and predictive behavior [184]. This layer is also the most uncovered and the most open because the application should be accessible on port 80 (HTTP) or port 443 (HTTPS) [185]. Due to these open ports, the device becomes vulnerable to cyberattacks such as security misconfiguration, SQL injection, and cross-site scripting [186]. The attacker can exploit any vulnerability present to deceive the user into controlling the application level [187].

10.7. Binder Transaction Redirection Attack

A binder is a fundamental component for Android applications that are used to access the admin controls and devices of the framework. It is also responsible for a client–server model, accepting that the system service is the server and the application is the client [188]. Thus, a binder transaction redirection (BiTRe) attack is launched to prompt system services for the smooth execution of a malicious server connection. This is because the connection is generally not monitored. Most Android system services can be isolated into three classes based on the kind of weakness being taken advantage of.

- The configuration vulnerability exploits authorization checks and access assets without application privileges.
- Information serialization for the Binder.
- Exploiting system service input validation to trigger memory corruption.

10.8. Active Warden Attack (AWA)

AWA is a method used to repackage an application by reverse engineering it and adding malicious functionality. This process is also known as "repackaging" and can be used to create a malicious version of a legitimate application [189]. The goal of an AWA is to evade detection using steganography to hide the malicious functionality within the app [189]. An attacker may use AWA to repackage and distribute an application to users, potentially stealing sensitive information or performing other malicious actions [190].

10.9. Android 3G Attacks

Android 3G attacks refer to various security vulnerabilities that exploit weaknesses in Android devices' 3G cellular network protocol. These vulnerabilities can allow an attacker to intercept and manipulate the data transmitted over the 3G network, potentially leading to the theft of sensitive information or unauthorized access to the device. These attacks have become increasingly prevalent with the widespread use of 3G capabilities on smartphones. Researchers have demonstrated a variety of attack scenarios that exploit these vulnerabilities, such as DoS attacks, international mobile subscriber identity (IMSI) paging attacks, and authentication key agreement (AKA) protocol attacks [191,192]. These attacks can expose mobile devices to monitoring and other cyber threats. Users need to keep their devices updated with the latest security patches to protect against these attacks and be aware of the potential risks associated with 3G connections.

10.10. Android 4G Attacks

Android 4G attacks refer to security vulnerabilities that exploit weaknesses in Android devices' 4G cellular network protocol. LTE, also known as 4G, is the fourth generation of cellular network technology that provides faster data transfer speeds and improved network capacity compared to 3G. As 4G networks have become more widespread, these attacks have become increasingly prevalent. Researchers have identified a variety of attack scenarios that exploit these vulnerabilities, such as location leakage attacks and DoS attacks [192,193]. These attacks can compromise the security of a device and its connected network. In location leakage attacks, an attacker can determine a device's location by exploiting vulnerabilities in the protocol used for location-based services on 4G networks. DOS attacks, on the other hand, can prevent legitimate users from accessing the network.

10.11. Android 5G Attacks

Android 5G attacks refer to security vulnerabilities that exploit weaknesses in Android devices' 5G cellular network protocol. The fifth generation of cellular network technology (5G), designed to provide faster data transfer speeds, lower latency, and improved network capacity compared to previous generations of cellular networks. Despite its many benefits, 5G networks lack a robust security posture. Researchers have demonstrated various attack scenarios that exploit these vulnerabilities, such as location tracking and call interception attacks [192]. These attacks can compromise the security of a device and its connected network. For example, an attacker with little knowledge of the broadcast paging protocols can launch a location-tracking attack by exploiting the vulnerabilities in the protocol used for location-based services on 5G networks. Similarly, call intercept attacks allow the attacker to intercept and listen to the call.

10.12. Android 6G Attacks

Android 6G attacks refer to security vulnerabilities that exploit weaknesses in Android devices' 6G cellular network protocol. The sixth generation of cellular network technology (6G), currently in development, is expected to provide even faster data transfer speeds, lower latency, and improved network capacity compared to 5G [194]. Although 6G technology is not yet widely available, researchers have already demonstrated the potential security threats. One example is a demonstration of eavesdropping on the 6G frequency using a DIY metasurface. This experiment, conducted by Z. Shaikhanov on 16 May 2022, a tool was developed using metallic foil, paper, a laminator, and inkjet printer components to eavesdrop on 6G wireless signals [195]. This successful demonstration highlights the need for more security enhancements to protect against potential attacks on 6G networks [196].

10.13. Quantum Threats to Android Smartphone

With advancements in technology, quantum computing became popular and practical to use in attack and defense scenarios. This technology poses a threat to the security and privacy of smartphones. This is because smartphones use public-key cryptography for authentication, communication, and encryption in various applications. On the other hand, Quantum computers' ability to generate factoring large numbers and simultaneously crack down renowned ciphers make them a persistent threat to public-key cryptosystems. For example, Android smartphones use Rivest-Shamir-Adleman (RSA), elliptic curve cryptography (ECC), and digital signature algorithm (DSA) for authentication, digital signatures, encryption, and key exchange [197]. They are responsible for ensuring the device's confidentiality, integrity, and authenticity. Therefore, Android smartphones are vulnerable to quantum attacks because of their features and characteristics. First, they use public-key cryptography for various security applications, such as securing web browsing, email communication, online banking, digital payments, VPN connections, Bluetooth pairing, Wi-Fi authentication, and device encryption. Second, they store and transmit sensitive data and credentials that could be valuable for attackers in present or future scenarios [198]. In the near future, a quantum-empowered attacker could decrypt the encrypted data and communications, forge digital signatures and certificates, and impersonate legitimate parties. This could lead to data breaches, identity theft, fraud, malware infection, and other cyberattacks [199–201].

Therefore, to stay ahead of these threats and mitigate their risk, it is crucial to take proactive measures and make Android resilient. One of the possible measures we can adopt is the usage and implementation of post-quantum cryptography (PQC). This involves building robust cryptographic algorithms that are resilient against quantum computing [202]. However, various challenges exist in implementing PQC resilient algorithms, such as limited memory, processing, and low battery life. Similarly, PQC implementations need to be developed for different architectures, such as Cortex-M4 and Cortex-A, and we will be required to accept trade-offs in terms of speed, memory, and power consumption [203]. However, this is essential, and by doing so, we will be able to ensure the confidentiality, integrity, and availability of smartphones.

11. Threat Detection and Mitigation

This section provides a comprehensive overview of the various detection mechanisms for Android malware attacks. It delves into the details of static and dynamic analysis techniques used to detect malware and outlines the key differences between them. This section also lists various Android malware detection tools that researchers have developed to improve security against malware attacks. Table 4 presents a detailed explanation of the Android malware detection tools developed between 2015 and 2023. These tools can be broadly classified into two main categories: static analysis and dynamic analysis. The table provides a detailed evaluation of each tool, including the year of its introduction, the methodology it uses, the dataset it employs, and whether it is open source. This information allows for comparing and contrasting the various tools and selecting the most suitable tool for a specific use case. It is also crucial to note that these tools can be combined to achieve better detection results by leveraging the strengths of each method.

Table 4. Comparative analysis of Android malware detection tools.

Name	Type	Year	Methodology	Discussion	Dataset	Open Source
CFSBFDroid [204]	Static analysis	2022	Performs detection using CFS and best first search on permissions and API calls	Groups up to seven algorithms (REPTree, RF, SMO, SGD, Rule PART, RF, and LMT)	Public dataset	No
S3Feature [205]	Static analysis	2022	This is a subgraph-based feature selection technique that uses a sensitive function call graph)	Can be an extension of existing malware detection tools and shows effective validation	Public dataset	No
ProDroid [206]	Static analysis	2021	Uses signatures, services, broadcasts, and API calls for detection	Cannot detect obfuscation and repackaged malware	Public dataset	No
DL-Droid [207]	Dynamic analysis	2020	Uses enhanced input generation and deep learning during sandbox analysis	Has a lack of diversity in the dataset that leads to poor validation and detection	30,000 apps	GitHub
SeqDroid [208]	Dynamic analysis	2019	Uses stacked convolutional and recurrent neural networks to detect obfuscated malware	Unable to detect APTs, dynamic code loading, and side-channel attacks	2,888,620 VirusTotal	No
DroidEvolver [209]	Static analysis	2019	Detection is based on API calls, runtime permissions, and signatures	Has limited feature set and lack of diversity in the dataset that leads to poor results	68,016 open source	GitHub
RoughDroid [210]	Dynamic analysis	2018	Uses a set of permissions, API calls, services, and intents	Unable to counter contemporary threats and side-channel attacks	Deribit dataset	No
MalDozer [211]	Dynamic analysis	2018	Uses deep learning algorithms on API calls and services	Lower accuracy due to the diversity of the dataset used for training and testing	71,000, Drebin, Virusshare	No
HinDroid [212]	Static analysis	2017	Uses a heterogeneous information network (HIN) on permission, intents, and API calls	Unable to detect APTs and side-channel attacks	32,334	No
DynaLog [213]	Dynamic analysis	2016	Classifies Android applications based on API calls and services executions	Lack of real-time data, and limited feature sets are deceivable	1940 open source APK	Github
ICCDetector [214]	Static analysis	2016	Analyzes application code, services, intents, and intent filters	Unable to detect malicious code loading and evolving malware families	17,290, open source APK	No
APK Auditor [215]	Static analysis	2015	Performs Android manifest permission-based malware classification	Insufficient training data, and limited feature sets are unable to defend against APTs	8762	No

Table 4. *Cont.*

Name	Type	Year	Methodology	Discussion	Dataset	Open Source
FlowDroid [216]	Static analysis	2014	Uses package name and methods call to perform analysis and workflow	Unable to detect hidden API calls, methods, and dynamic code loading	1500	Github
vetDroid [217]	Static analysis	2013	Reconstructs sensitive behaviors in Android apps from a permission-based detection	Fails to identify APTs, sensor-based attacks, and side-channel attacks	1249 apps	No
DroidMat [218]	Static analysis	2012	Decompiles and dissolves applications and use dex code to extract suspicious API calls	Unable to detect obfuscation, dynamic code loading, and contemporary threats	1738 apps	No
Droidbox [219]	Static analysis	2011	Analyzes network traffic to identify security and privacy threats to android users	Unable to detect dynamic code loading, side-channel attacks, and sensor-based attacks	Not disclosed	Github
AASandbox [220]	Dynamic analysis	2010	Performs both static and dynamic analysis to identify malicious patterns and application logs	Unable to detect contemporary threats, Google Play attacks, and side-channel attacks	Not disclosed	No
Scandroid [220]	Static analysis	2009	Applies data flow analysis on APK files to detect suspicious patterns	Unable to detect side-channel attacks, Google Play attacks, and contemporary threats	Not disclosed	No

11.1. Static Malware Analysis

Static malware analysis in Android is a technique used to analyze the code and structure of an Android application without executing it. This technique aims to detect a malicious code or behavior within the application that could potentially harm the device or its user [221]. Static analysis is performed without running a malicious file, and it finds the malicious applications using various types of information that can be obtained when the application is reversed. This includes malware signatures, application permissions, hardcoded strings, protocols, Dalvik bytecode, function information, opcode sequence, control flow graphs (CFGs), and other types of information [222]. By using these different types of information, analysis can identify patterns or characteristics commonly found in malware and flag an application as potentially malicious. In some cases, manual code review is also an important aspect of static analysis, in which a security researcher manually examines the code for signs of malicious behavior or vulnerabilities [108]. This can include looking for specific strings or code patterns known to be associated with malware or analyzing the permissions and intents used by the application [223].

1. *Signature-based detection:* This technique involves identifying malware by searching for known patterns or "signatures" in the code of an application. This can be used to detect known malware families or variants.
2. *Permission-based detection:* This technique involves identifying malware by analyzing the permissions requested by an application. Malicious applications may request permissions that are not required for their intended function or are unusual for the application category.
3. *Code analysis:* This technique involves analyzing the code of an application to identify malicious behavior. This can be performed by manually examining the code or using automated tools to generate control flow graphs, data flow diagrams, and other code representations.
4. *String analysis:* This technique involves identifying malware by searching for hard-coded strings, such as URLs, IP addresses, or file paths, in the code of an application. This can detect command-and-control servers, domains, or other infrastructure used by malware.
5. *Opcode analysis:* This technique involves the identification of malware by analyzing the opcode sequences in the code of an application. This can detect malware that uses specific code sequences or instructions, such as those used by known malware families.
6. *Bytecode analysis:* This technique involves identifying malware by analyzing the Dalvik bytecode of an application. This can detect malware that uses specific bytecode sequences or instructions, such as those used by known malware families.
7. *Resource analysis:* This technique involves identifying malware by analyzing an APK's resources. This can be used to detect malware that uses specific resources, such as images or audio files, that are not required for the intended function of the application.
8. *Manifest analysis:* This technique involves identifying malware by analyzing the AndroidManifest.xml file of an APK. This can detect malware that uses specific manifest attributes, such as permissions, broadcast receivers, and threads, such as those used by known malware families.

11.2. Dynamic Malware Analysis

Dynamic malware analysis on Android is a technique used to analyze the behavior of an Android application while running [224]. This technique executes the malicious APK in a controlled and monitored environment such as an application sandbox, emulators, or virtual machines [225]. This allows for capturing a wide range of data that can be analyzed for signs of malicious activity.

Dynamic malware analysis aims to detect malicious behavior or vulnerabilities within the application that could potentially harm the device or its user [226]. This technique uses various detection methods to identify malicious activity in an application. Some of the common methods used are:

1. *System call monitoring:* This allows for the capture of system calls made by the application, which can provide insights into the system resources that the application is accessing and can be used to detect malicious behavior or vulnerabilities.
2. *Runtime behavior:* This includes monitoring the application's behavior while it is running. This can detect malicious behavior, such as attempts to exfiltrate data or to gain unauthorized access to system resources.
3. *Device traces:* This includes capturing information about the device, such as location data, call logs, and contacts. These data can be used to detect attempts to steal personal information or track a device's location.
4. *API calls:* This involves monitoring the application's application programming interfaces (APIs) to detect attempts to access restricted resources or perform malicious actions.
5. *Registry changes:* This involves monitoring the changes made to the system registry by the application, which can be used to detect attempts to install malicious software or to make unauthorized changes to the system.
6. *Memory writes:* This involves monitoring the writes to the memory by the application, which can be used to detect attempts to inject malware or to execute code in a privileged context.
7. *Network traffic monitoring:* This involves monitoring the network traffic generated by the application, which can detect attempts to exfiltrate data or communicate with command-and-control servers.
8. *Code instrumentation:* This technique involves modifying the original code of an application to insert hooks or probes at specific points of interest. This allows for the collection of detailed information about the application's behavior.

11.3. Fault Detection and PQC Implementations

Post-quantum cryptography (PQC) is paramount in smart devices, where long-term security is key to sustainability. Critical operations such as application signing, keystore generation, key exchange, and digital signatures improvise robust PQC implementations [227]. Meanwhile, the role of fault detection in ensuring the accuracy and dependability of cryptographic implementations is crucial. Therefore, several fault detection methods are employed to guarantee the integrity and reliability of cryptographic algorithms. These methods serve to identify any abnormalities and ensure the consistency of results. By leveraging these fault detection techniques, potential anomalies can be detected. Fault detection mechanisms are intricately integrated within PQC implementations that encompass a range of techniques tailored to each cryptographic algorithm, ensuring the thorough scrutiny and verification of every step in the process [228,229]. Details are outlined as follows.

11.3.1. Curve448 and Ed448 on Cortex-M4

The Cortex-M4 is a 32-bit microcontroller widely utilized in smartphones, embedded systems, and IoT devices. It offers robust computation capabilities, including ALU, CPU frequency, RAM, and ROM, surpassing traditional processors [230].

Curve448 and Ed448 are elliptic curves explicitly designed for the ECDH key agreement scheme and the EdDSA digital signature algorithm. These curves provide a high level of security with 224-bit strength and optimize the implementation of Curve448 and Ed448 on Cortex-M4, which involves carefully optimizing finite-field arithmetic and group operations utilized in the protocols [231]. Moreover, techniques such as Karatsuba multiplication, Montgomery reduction, lazy reduction, and conditional swaps play a significant role in achieving efficient computation on the Cortex-M4. Similarly, tailoring existing methods to leverage the microcontroller's features, such as bit manipulation instructions, barrel shifter, and constant-time execution, proves effective in enhancing key agreement schemes [232]. Finally, by carefully optimizing the finite-field arithmetic and group operations while leveraging the specific features of the Cortex-M4, the implementation of Curve448 and Ed448 on this microcontroller platform can deliver efficient and secure cryptographic operations for applications involving ECDH and EdDSA protocols.

11.3.2. SIKE on Cortex-M4

Supersingular isogeny key encapsulation (SIKE) protocol utilizes the Diffie–Hellman key exchange protocol based on arithmetic operations on elliptic curves and isogeny maps for secure communication. To enhance security and privacy, the implementation focuses on constant-time and constant-memory algorithms that prevent information leakage through side channels [230]. Moreover, additional measures such as error correction codes, redundancy checks, and masking techniques are applied to detect and correct faults, further ensuring the system's integrity. Finally, to evaluate and improve the security posture, the implementation undergoes fault injection attacks to test its robustness and identify vulnerabilities [233]. This rigorous testing helps strengthen the overall security of the SIKE implementation on Cortex-M4, ensuring its resilience against potential attacks.

11.3.3. SIKE Round 3 on ARM Cortex-M4

SIKE Round 3 is an optimized implementation of the SIKE protocol specifically designed for low-power microcontrollers. It leverages the unique features of the ARM Cortex-M4 architecture to achieve improved performance and reduced memory footprint [234]. With the smallest public key and ciphertext sizes among NIST candidates, it offers efficient post-quantum secure communication for resource-constrained devices. It has some advantages, such as a significantly improved SIKE protocol performance, allowing for faster key encapsulation and decapsulation operations on resource-constrained devices [235]. Additionally, the compactness of the SIKE Round 3 implementation on the ARM Cortex-M4 results in a reduced memory footprint. The implementation takes advantage of the specific features and instruction set of the ARM Cortex-M4 processor to achieve better performance [236].

11.3.4. Kyber on 64-Bit ARM Cortex-A

Kyber on 64-bit ARM Cortex-A represents a robust implementation of the Kyber post-quantum key encapsulation mechanism meticulously designed for the powerful ARM Cortex-A microarchitecture. Leveraging the exceptional computational capabilities inherent to Cortex-A processors, this optimized implementation delivers accelerated key encapsulation and decapsulation operations, significantly reducing latency and heightened overall efficiency [237]. With its advanced features, including larger registers, more expansive SIMD units, and elevated memory bandwidth, Kyber on Cortex-A exhibits unparalleled performance, minimal latency, and exceptional efficiency, making it an ideal choice for a wide array of sophisticated applications spanning mobile devices, embedded systems, and high-performance servers [238]. Employing a professional-grade design methodology, this implementation ensures utmost security and reliability, fortifying cryptographic capabilities in the realm of post-quantum key encapsulation. Its seamless integration and scalability further augment its appeal, rendering Kyber on 64-bit ARM Cortex-A an indispensable solution for demanding cryptographic requirements in professional settings.

11.3.5. Cryptographic Accelerators on Ed25519

Cryptographic accelerators for Ed25519 are specialized hardware components designed to optimize and accelerate the cryptographic operations associated with the Ed25519 elliptic curve digital signature algorithm [239]. Ed25519 is widely recognized for its efficiency and robust security properties. The implementation of cryptographic accelerators for Ed25519 aims to offload computationally intensive tasks to dedicated hardware, resulting in substantial performance improvements. These accelerators are meticulously designed to efficiently handle the finite-field arithmetic and elliptic curve operations essential for the Ed25519 algorithm [240]. By leveraging dedicated hardware or specialized accelerators, the cryptographic functions involved in Ed25519, including key generation, signing, and verification, can be executed significantly faster than software-based implementations. This dramatic increase in speed and efficiency empowers applications that rely on Ed25519 for high-performance digital signatures.

In addition to enhanced performance, cryptographic accelerators for Ed25519 often incorporate advanced security features, such as robust protection against side-channel attacks and tampering. These safeguards fortify the security posture of the cryptographic keys and prevent the leakage of sensitive information through potential side-channel vulnerabilities [241]. Overall, using cryptographic accelerators for Ed25519 enables the swift and secure execution of the algorithm's operations. By optimizing performance and efficiency, these accelerators elevate Ed25519's suitability for various applications, including secure communication protocols, cryptographic authentication mechanisms, and secure code signing. Their professional-grade design and integration provide a trusted and dependable solution for robust digital signature requirements.

11.4. Lightweight Ciphers Fault Detection

Fault attacks target lightweight ciphers by intentionally introducing faults during their execution of the algorithm. The aim is to exploit vulnerabilities and extract sensitive information [242,243]. These attacks are targeted to breach the security and integrity of cipher. Therefore, multiple techniques are used to defend against the risk of fault attacks, which empower the security and integrity of the cipher and make it resilient. The details are outlined below.

11.4.1. Fault Detection of Architectures of Pomaranch Cipher

The Pomaranch cipher is a lightweight stream cipher that is designed for low-power devices. It is vulnerable to fault detection attacks; therefore, it is important to implement security measures to identify and mitigate them. To achieve this, redundancy-based methods can be used that will be introduced by duplicating critical components [243]. For example, using error-detection codes, we can identify parity bits and calculate checksums to quickly identify the fault attacks and mitigate their impact [244].

11.4.2. Reliable Architectures of Grostl Hash

Grostl hash is a cryptographic function specifically designed for resource-constrained environments. Although it has reliable architectures that somewhat resist fault attacks, it still proves ineffective and resilient [245]. Therefore, it is important to ensure security and privacy by properly detecting and mitigating fault attacks. To achieve this, the modular redundancy method can be used. The architecture of the hash function can be intentionally designed with more than one redundant module that independently performs computations [246]. Finally, in the end, we can compare the outputs of these modules to detect faults if there is a difference in output. Afterwards, the fault can be identified and rectified.

11.4.3. Fault Diagnosis of Low-Energy Midori Cipher

The low-energy Midori cipher is a lightweight block cipher mostly used in low-power devices. The cipher is fast, uses less energy, and has low latency. Anita et al. [247] provided an effective scheme for fault detection in the Midori cipher. They provided scheme work on nonlinear S-box layers for both 128-bit and 64-bit architectures. Researchers used the LUT-based implementation of logic gates that worked in signature-based error detection. The technique proposed is effective and significantly contributes to the reliability of the cipher.

11.4.4. Fault Diagnosis of RECTANGLE Cipher

RECTANGLE is a lightweight block cipher that uses an SP-network structure with 25 rounds and a 64-bit block size. The cipher is designed for both hardware and software implementations that use bit-slice techniques, and it has a key size of 128-bit [248]. Due to the cipher's significant importance, detecting faults at different cipher levels, such as the S-box layer, the P-layer, or the round structure, is crucial. Nallathambi et al. [247] proposed a detection scheme that can effectively detect faults in the RECTANGLE cipher. The authors used parity checking for the S-box layer and CRC for the P-layer and the round structure. They also used a fault counter to monitor the detected faults and trigger an alarm if it exceeds a threshold. They benchmark their scheme on FPGA and show that it is close to 100.

12. Open Issues and Challenges

This section details open issues, challenges, and the latest standards for the Android operating system. This section discusses several prevalent issues, including version fragmentation, screen overlay, and dynamic code loading, as outlined below.

12.1. Version Fragmentation

Version fragmentation in Android refers to the phenomenon in which different devices are running different versions of the Android operating system. This means that not all devices are running the latest version of Android, and some are running older versions [249,250]. This can create problems for app developers, as they need to ensure that their apps are compatible with a wide range of different versions of Android. Furthermore, it can also create security vulnerabilities, as older versions of Android may have a different levels of security than the latest version [251]. Several factors contribute to the fragmentation of Android versions, including the slow pace of updates from device manufacturers, many different Android devices on the market, and the fact that some devices are no longer supported by their manufacturers [252]. This can make it difficult for users and developers to ensure that their devices and apps are running the latest version of the Android. One big reason behind version fragmentation is that many mobile device manufacturers are dealing with various versions of the Android as they have control over when their customers receive updates and limit updates to specific devices that run the latest operating system. For instance, a device manufacturer may only provide updates for Android version 11 or higher devices. This version fragmentation is a major problem in the Android ecosystem, as it results in a significant portion of devices remaining vulnerable to threats that have already been addressed by either Google or device manufacturers through patches and updates.

12.2. Privacy Risks of Third-Party Applications and Libraries

The privacy risks associated with third-party applications developed by nontrusted providers and open application stores are an open challenge for device manufacturers and security professionals. Some of these applications collect and share personal data from users without their knowledge or consent to generate revenue through targeted advertising [253]. These applications use third-party libraries, such as Google Analytics, Firebase, and Facebook Analytics, to collect user demographics, interests, age, gender, and preferences to show users more relevant ads. These libraries are often integrated into the mobile application using a software development kit (SDK) provided by the data collection service. The data collected can include information such as device type, device identifier, IP address, location, app usage data, and browsing history. These data are then shared with the data collection service, which uses them to perform targeted advertising and create user profiles [254].

However, using third-party libraries can introduce security vulnerabilities in the mobile application. For example, if the data collection service's servers are hacked, the user data can be exposed. Additionally, if the mobile application is not properly configured, the data collection service may be able to access sensitive information such as contacts, photos, and microphone recordings [255]. The app developers may also use these data to deceive the user by showing them unwanted ads or even sharing the data with other third parties, which is against the user's privacy. Furthermore, as discussed in previous sections, many developers successfully demonstrated that the Google Play Store is vulnerable to potential cyber threats. Moreover, Google Play Store's requirement for a privacy policy can be deceived by mobile application developers, and the fact that many free platforms create privacy policies with simple clicks fails to detect these kinds of privacy breaches.

This can put users at risk of serious privacy breaches, including identity theft and financial fraud [256–258].

Furthermore, federated learning can be used to improve the privacy of smartphone data [259–262]. Federated learning works by creating a server model and any number of clients [263]. Federated learning keeps the local data to local clients only. Only the parameters of models are shared with the global model.

12.3. Dynamic Code Loading

Dynamic code loading refers to an application's ability to load and execute code at runtime rather than at the time of installation [264]. This is often used to update an app or load new features without requiring the user to manually update the app. However, this can also introduce security risks as it may allow malicious code to be loaded and executed on a user's device. Dynamic code loading is an open issue for the Android operating system because attackers can exploit it to gain unauthorized access to a user's device or data [108,265]. For example, an attacker may use dynamic code loading to load malware onto a user's device or steal sensitive information. Additionally, dynamic code loading can be used to evade detection by the security software, making it harder for users to protect themselves from malicious apps. The sole purpose of dynamic code loading is to achieve legitimate functionality, such as runtime updates and application size reduction, but attackers exploit this vulnerability for runtime malicious code execution. Attackers are mostly successful in deciding targets and performing malicious activities because runtime-loaded code libraries are not accessible by Google Play Protect or any installed antivirus applications. That is why remote code can be used to bypass anti-malware solutions.

12.4. Limited Traceability and Data-Theft Protection

Protecting personal data and device traceability in Android devices that are lost or stolen is a critical issue in today's digital landscape. With the increasing use of mobile devices for personal and professional purposes, the security of these devices has become a primary concern. Unfortunately, Android devices are particularly vulnerable to theft and loss, which can expose sensitive information, such as personal contacts, photos, and financial data. One of the main challenges in addressing this issue is the need for built-in tracking and remote wiping capabilities on many Android devices. Although some manufacturers have added these features to their devices, they are only sometimes enabled by default and may only be available on some devices. This makes it difficult for users to locate lost or stolen devices and protect their data. Many users turn to third-party antitheft applications to mitigate this risk, such as Google's 'Find My Device', Cerberus, McAfee Mobile Security, and Crook Catcher. These applications can provide some level of protection, such as the ability to locate a lost or stolen device and remotely wipe its data. However, these applications are limited in capabilities and may not provide a comprehensive solution for protecting personal data.

12.5. NIST Lightweight Cryptography Standardization

NIST realized the need and empowered a secure and efficient cryptographic algorithm for Android devices that works on limited resources, such as memory, power, or bandwidth. Initially initiated in 2018, the process received 57 submissions [266]. In February 2023, NIST announced the selection of the Ascon family as the winner of the lightweight cryptography standardization process.

Ascon is a group of cryptographic algorithms based on a sponge construction and uses a permutation function with a 320-bit state [267]. It provides three variants of AEAD algorithms (Ascon-128, Ascon-128a, and Ascon-80pq) and one variant of a hash algorithm (Ascon-Hash). Ascon is designed to be secure against quantum attacks and to have low energy consumption and high performance on various platforms, including Android devices.

This standard will help secure Android devices by providing a powerful and efficient way to encrypt and authenticate data created and transmitted by them. Using Ascon algorithms, Android devices can protect their data from unauthorized access, modification, or tampering by adversaries who may have access to quantum computers or physical attacks. Ascon algorithms can also improve Android devices' performance and battery life by reducing computational complexity and the energy consumption of cryptographic operations.

13. Conclusions and Future Work

In this article, we conducted a comprehensive survey of the Android ecosystem. It summarizes key contributions from 2009 to 2023 and provides a comprehensive article highlighting Android vulnerabilities, cyber threats, and ways to mitigate them. It also details current security and privacy challenges in Android and critical issues that need to be addressed. We compared our work with existing research articles across ten different characteristics to validate our findings. Our key contributions include, but are not limited to, a detailed survey of Android software and hardware vulnerabilities. Additionally, the article includes APTs, side-channel attacks, cryptographic attacks, power fault attacks, and attacks launched through the Google Play Store. Moreover, to mitigate these threats, a detailed study on mobile threat defense has been conducted so that a comparison of tools is available to detect and secure Android devices. Finally, we highlighted current open issues and possible research directions, such as version fragmentation, the privacy risks of third-party applications, and dynamic code loading. Finally, the subsequent subsection provides details of possible future work and research directions.

Future Work

With the current technological advancements, mobile technology faces numerous challenges that must be overcome. Research contributions are required to counter challenges and develop defensive measures in the underlined directions:

- *Android version fragmentation and dynamic code loading threats:* Future work should address the challenges of Android version fragmentation, which affects app compatibility and update delivery. Similarly, dynamic code loading is a leading threat exploited for malicious purposes which can be significantly reduced using Project Treble architecture and enforcing the Play system updates only.
- *Post-quantum cryptography for Android:* Future work should address building PQC resilient cryptographic algorithms, protocols, and authentication mechanisms for Android devices.
- *Fault attacks and power analysis attacks for Android:* Future work should focus on developing and implementing fault detection and power analysis-resistant software and hardware to prevent these side-channel attacks.
- *Mitigation of malware propagation on application stores:* Future work should focus on developing and implementing mitigation techniques for malware and APT propagation of Google Play Store and other third-party application stores.
- Finally, comprehensive research and development can be conducted to mitigate or minimize the effect of all the discussed attacks and open issues such as active warden attacks, standHogg attacks, keystore forgery attacks, and application collision attacks.

This article serves as a valuable resource for security researchers, Android developers, and industry experts to use this knowledge base. It provides numerous mitigation techniques to respond to the threats discussed. The research invites academics, developers, security researchers, and industry experts to contribute towards developing security tools and further improve the discussed open issues for the security of future-generation smartphones.

Author Contributions: Z.M.; methodology, Z.M., A.R.J., T.R.G., Z.A. and S.A.; validation, A.R.J. B.S. and Z.A.; formal analysis, Z.M., S.A., A.R.J., Z.A., T.R.G. and B.S.; writing—original draft preparation, Z.M., B.S., T.R.G., S.A.; writing—review and editing, A.R.J., Z.A., T.R.G. and A.R.J.; supervision, S.A. and A.R.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Sheila and Robert Challey Institute for Global Innovation and Growth, North Dakota State University (NDSU), USA.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest

References

1. Herrero, J.; Rodríguez, F.J.; Urueña, A. Use of smartphone apps for mobile communication and social digital pressure: A longitudinal panel study. *Technol. Forecast. Soc. Chang.* **2023**, *188*, 122292. [CrossRef]
2. Khan, L.U.; Saad, W.; Han, Z.; Hossain, E.; Hong, C.S. Federated learning for internet of things: Recent advances, taxonomy, and open challenges. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1759–1799. [CrossRef]
3. Kaur, P.; Arora, K. Internet of Things-Based Economical Smart Home Automation System. In *Industrial Internet of Things*; CRC Press: Boca Raton, FL, USA, 2022; pp. 129–142.
4. Toppo, P.; Dhote, T. Preference of Mobile Platforms: A Study of Ios vs. Android. *Int. J. Mod. Agric.* **2021**, *10*, 1757–1764.
5. Analytica, O. Huawei's Harmony may challenge Android-Apple duopoly. *Emerald Expert Briefings* **2021** . [CrossRef]
6. Garg, S.; Baliyan, N. Comparative analysis of Android and iOS from security viewpoint. *Comput. Sci. Rev.* **2021**, *40*, 100372. [CrossRef]
7. Statista, J. Smartphone OS Market Share Forecast 2014–2023, 2022.
8. Chawla, A. Pegasus Spyware—'A Privacy Killer'. *SSRN* **2021**. [CrossRef]
9. Thomas, T.; Surendran, R.; John, T.S.; Alazab, M. *Intelligent Mobile Malware Detection*; CRC Press Routledge Publisher: Boca Raton, FL, USA, 2022.
10. Jabar, T.; Mahinderjit Singh, M. Exploration of Mobile Device Behavior for Mitigating Advanced Persistent Threats (APT): A Systematic Literature Review and Conceptual Framework. *Sensors* **2022**, *22*, 4662. [CrossRef]
11. Acharya, S.; Rawat, U.; Bhatnagar, R. A Comprehensive Review of Android Security: Threats, Vulnerabilities, Malware Detection, and Analysis. *Secur. Commun. Netw.* **2022**, *2022*, 7775917. [CrossRef]
12. Kady, C.; Chedid, A.M.; Kortbawi, I.; Yaacoub, C.; Akl, A.; Daclin, N.; Trousset, F.; Pfister, F.; Zacharewicz, G. Iot-driven workflows for risk management and control of beehives. *Diversity* **2021**, *13*, 296. [CrossRef]
13. Chandrashekar, A.; Kumar, P.V.; Chandavarkar, B. Comparative Analysis of Modern Mobile Operating Systems. In Proceedings of the 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 6–8 July 2021; pp. 1–7.
14. Mahor, V.; Pachlasiya, K.; Garg, B.; Chouhan, M.; Telang, S.; Rawat, R. Mobile Operating System (Android) Vulnerability Analysis Using Machine Learning. In Proceedings of the International Conference on Network Security and Blockchain Technology, Huaihua City, China, 15–17 July 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 159–169.
15. Senanayake, J.; Kalutarage, H.; Al-Kadri, M.O.; Petrovski, A.; Piras, L. Android source code vulnerability detection: A systematic literature review. *ACM Comput. Surv.* **2023**, *55*, 1–37. [CrossRef]
16. Javed, A.R.; Beg, M.O.; Asim, M.; Baker, T.; Al-Bayatti, A.H. Alphalogger: Detecting motion-based side-channel attack using smartphone keystrokes. *J. Ambient. Intell. Humaniz. Comput.* **2023**, *14*, 4869–4882. [CrossRef]
17. Javed, A.R.; Rehman, S.U.; Khan, M.U.; Alazab, M.; Khan, H.U. Betalogger: Smartphone sensor-based side-channel attack detection and text inference using language modeling and dense multilayer neural network. *Trans. Asian Low-Resour. Lang. Inf. Process.* **2021**, *20*, 1–17. [CrossRef]
18. Muhammad, Z.; Amjad, F.; Iqbal, Z.; Javed, A.R.; Gadekallu, T.R. Circumventing Google Play vetting policies: A stealthy cyberattack that uses incremental updates to breach privacy. *J. Ambient. Intell. Humaniz. Comput.* **2023**, *14*, 4785–4794. [CrossRef]
19. Deeban Chakravarthy, V.; Prakash, K.L.; Ramana, K.; Gadekallu, T.R. A Novel DDOS Attack Detection and Prevention Using DSA-DPI Method. In Proceedings of the International Conference on Innovative Computing and Communications: Proceedings of ICICC 2022, Delhi, India, 18–19 November 2022; Springer: Berlin/Heidelberg, Germany, 2022; Volume 3, pp. 733–743.
20. Wang, D.; Chen, T.; Zhang, Z.; Zhang, N. A Survey of Android Malware Detection Based on Deep Learning. In Proceedings of the International Conference on Machine Learning for Cyber Security, Nadi, Fiji, 2–4 December 2023; Springer: Berlin/Heidelberg, Germany, 2023; pp. 228–242.
21. Meijin, L.; Zhiyang, F.; Junfeng, W.; Luyu, C.; Qi, Z.; Tao, Y.; Yinwei, W.; Jiaxuan, G. A Systematic Overview of Android Malware Detection. *Appl. Artif. Intell.* **2022**, *36*, 2007327. [CrossRef]
22. Saab, S.S.; Shen, D.; Orabi, M.; Kors, D.; Jaafar, R.H. Iterative learning control: Practical implementation and automation. *IEEE Trans. Ind. Electron.* **2021**, *69*, 1858–1866. [CrossRef]
23. Cao, M. Understanding the characteristics of invasive malware from the Google Play Store. Ph.D. Thesis, University of British Columbia, Vancouver, BC, Canada , 2022.
24. Wang, X. Security Threats and Protection Based on Android Platform. In Proceedings of the 2021 International Conference on Big Data Analytics for Cyber-Physical System in Smart City, Bangkok, Thailand, 16–17 December 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 179–186.

25. Muhammad, Z.; Amjad, M.F.; Abbas, H.; Iqbal, Z.; Azhar, A.; Yasin, A.; Iesar, H. A Systematic Evaluation of Android Anti-Malware Tools for Detection of Contemporary Malware. In Proceedings of the 2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC), Shenyang, China, 20–22 October 2021; pp. 117–124.
26. Cheng, B.; Kikuta, T.; Toshimitsu, Y.; Saito, T. Investigation of Power Consumption Attack on Android Devices. In Proceedings of the International Conference on Advanced Information Networking and Applications, Toronto, ON, Canada, 12–14 May 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 567–579.
27. Wu, H.; Zhang, H.; Wang, Y.; Rountev, A. Sentinel: Generating GUI tests for sensor leaks in Android and Android wear apps. *Softw. Qual. J.* **2020**, *28*, 335–367. [CrossRef]
28. Sikder, A.K.; Aksu, H.; Uluagac, A.S. A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Trans. Mob. Comput.* **2019**, *19*, 245–261. [CrossRef]
29. Dini, G.; Martinelli, F.; Matteucci, I.; Petrocchi, M.; Saracino, A.; Sgandurra, D. Risk analysis of Android applications: A user-centric solution. *Future Gener. Comput. Syst.* **2018**, *80*, 505–518. [CrossRef]
30. Hur, J.B.; Shamsi, J.A. A survey on security issues, vulnerabilities and attacks in Android based smartphone. In Proceedings of the 2017 International Conference on Information and Communication Technologies (ICICT), Karachi, Pakistan, 30–31 December 2017; pp. 40–46.
31. Xu, M.; Song, C.; Ji, Y.; Shih, M.W.; Lu, K.; Zheng, C.; Duan, R.; Jang, Y.; Lee, B.; Qian, C.; et al. Toward engineering a secure android ecosystem: A survey of existing techniques. *ACM Comput. Surv.* **2016**, *49*, 1–47. [CrossRef]
32. Tan, D.J.; Chua, T.W.; Thing, V.L. Securing android: A survey, taxonomy, and challenges. *ACM Comput. Surv.* **2015**, *47*, 1–45.
33. Faruki, P.; Bharmal, A.; Laxmi, V.; Ganmoor, V.; Gaur, M.S.; Conti, M.; Rajarajan, M. Android security: A survey of issues, malware penetration, and defenses. *IEEE Commun. Surv. Tutor.* **2014**, *17*, 998–1022. [CrossRef]
34. Wang, Y.; Zheng, J.; Sun, C.; Mukkamala, S. Quantitative security risk assessment of android permissions and applications. In *Proceedings of the Data and Applications Security and Privacy XXVII: 27th Annual IFIP WG 11.3 Conference, DBSec 2013, Newark, NJ, USA, 15–17 July 2013*; Proceedings 27; Springer: Berlin/Heidelberg, Germany, 2013; pp. 226–241.
35. La Polla, M.; Martinelli, F.; Sgandurra, D. A survey on security for mobile devices. *IEEE Commun. Surv. Tutor.* **2012**, *15*, 446–471. [CrossRef]
36. Becher, M.; Freiling, F.C.; Hoffmann, J.; Holz, T.; Uellenbeck, S.; Wolf, C. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In Proceedings of the 2011 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 22–25 May 2011; pp. 96–111.
37. Goode, A. Managing mobile security: How are we doing? *Netw. Secur.* **2010**, *2010*, 12–15. [CrossRef]
38. Maker, F.; Chan, Y.H. *A Survey on Android vs. Linux*; University of California: Los Angeles, CA, USA , 2009; pp. 1–10.
39. Chaudhary, A.; Gupta, H.P.; Shukla, K. Real-Time Activities of Daily Living Recognition Under Long-Tailed Class Distribution. *IEEE Trans. Emerg. Top. Comput. Intell.* **2022**, *6*, 740–750 . [CrossRef]
40. Jiang, X.; Liu, M.; Yang, K.; Liu, Y.; Wang, R. A security sandbox approach of android based on hook mechanism. *Secur. Commun. Netw.* **2018**, *2018* . [CrossRef]
41. Shabtai, A.; Fledel, Y.; Elovici, Y. Securing Android-powered mobile devices using SELinux. *IEEE Secur. Priv.* **2009**, *8*, 36–44. [CrossRef]
42. Garg, S.; Baliyan, N. Android security assessment: A review, taxonomy and research gap study. *Comput. Secur.* **2021**, *100*, 102087. [CrossRef]
43. Fatima, M.; Abbas, H.; Yaqoob, T.; Shafqat, N.; Ahmad, Z.; Zeeshan, R.; Muhammad, Z.; Rana, T.; Mussiraliyeva, S. A survey on common criteria (CC) evaluating schemes for security assessment of IT products. *PeerJ Comput. Sci.* **2021**, *7*, e701. [CrossRef]
44. Gupta, B.B.; Gaurav, A.; Marín, E.C.; Alhalabi, W. Novel graph-based machine learning technique to secure smart vehicles in intelligent transportation systems. *IEEE Trans. Intell. Transp. Syst.* **2022**. [CrossRef]
45. Muhammad, Z.; Anwar, Z.; Saleem, B.; Shahid, J. Emerging Cybersecurity and Privacy Threats to Electric Vehicles and Their Impact on Human and Environmental Sustainability. *Energies* **2023**, *16*, 1113. [CrossRef]
46. Chetan, R.; Avinash, N.; Aditya, K.; Gowri, M.; Pranav, K.; Namana. Providing Knee Movement Assistance using Android and IOT. In Proceedings of the 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 7–9 October 2021; pp. 140–145.
47. Hou, Q.; Diao, W.; Wang, Y.; Liu, X.; Liu, S.; Ying, L.; Guo, S.; Li, Y.; Nie, M.; Duan, H. Large-scale Security Measurements on the Android Firmware Ecosystem. In Proceedings of the International Conference on Software Engineering (ICSE'22), Pittsburgh, PA, USA, 21–29 May 2022; Association for Computing Machinery: New York, NY, USA, 2022.
48. Moulahi, T.; Jabbar, R.; Alabdulatif, A.; Abbas, S.; El Khediri, S.; Zidi, S.; Rizwan, M. Privacy-preserving federated learning cyber-threat detection for intelligent transport systems with blockchain-based security. *Expert Syst.* **2023**, *40*, e13103. [CrossRef]
49. Radhika, B.; Kumar, N.N.; Shyamasundar, R.; Vyas, P. Consistency analysis and flow secure enforcement of selinux policies. *Comput. Secur.* **2020**, *94*, 101816. [CrossRef]
50. Mayrhofer, R.; Stoep, J.V.; Brubaker, C.; Kralevich, N. The android platform security model. *ACM Trans. Priv. Secur.* **2021**, *24*, 1–35. [CrossRef]
51. Hutchinson, S.; Zhou, B.; Karabiyik, U. Are we really protected? An investigation into the play protect service. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 4997–5004.
52. Google. Protect against security threats with safetynet: Android developers.

53. Muhammad, Z.; Anwar, Z.; Saleem, B. A cybersecurity risk assessment of electric vehicle mobile applications: Findings and recommendations. In Proceedings of the 2023 3rd International Conference on Artificial Intelligence (ICAI), Wuhan, China, 17–19 November 2023; pp. 45–51.
54. Ning, P. Samsung knox and enterprise mobile security. In Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, Scottsdale, AZ, USA, 3–7 November 2014; Association for Computing Machinery: New York, NY, USA 2014; p. 1.
55. Le, T.D.B.; Bao, L.; Lo, D.; Gao, D.; Li, L. Towards mining comprehensive android sandboxes. In Proceedings of the 2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), Melbourne, Australia, 12–14 December 2018; pp. 51–60.
56. Brahler, S. Analysis of the android architecture. *Karlsr. Inst. Technol.* **2010**, *7*.
57. Framework, A.; Runtime, A.; Kernel, L. Android Architecture. *Android Developers*.
58. Farooqi, S.; Feal, Á.; Lauinger, T.; McCoy, D.; Shafiq, Z.; Vallina-Rodriguez, N. Understanding incentivized mobile app installs on google play store. In Proceedings of the ACM Internet Measurement Conference, Virtual, 27–29 October 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 696–709.
59. Suleman, M.; Soomro, T.R.; Ghazal, T.M.; Alshurideh, M. Combating Against Potentially Harmful Mobile Apps. In Proceedings of the The International Conference on Artificial Intelligence and Computer Vision, Settat, Morocco, 28–30 June 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 154–173.
60. Kumar, S.; Shanker, R.; Verma, S. Context aware dynamic permission model: A retrospect of privacy and security in android system. In Proceedings of the 2018 International Conference on Intelligent Circuits and Systems (ICICS), Phagwara, India, 20–21 April 2018; pp. 324–329.
61. Alkindi, Z.R.; Unviresity, S.Q.; Muscat, O.; Sarrab, M.; Alzidi, N. Android Application Permission Model. In Proceedings of the 4th Free & Open Source Software Conference (FOSSC'2019-OMAN), Abu Dhabi, Muscat, 11–12 February 2019.
62. Zhan, X.; Liu, T.; Fan, L.; Li, L.; Chen, S.; Luo, X.; Liu, Y. Research on Third-Party Libraries in Android Apps: A Taxonomy and Systematic Literature Review. *IEEE Trans. Softw. Eng.* **2021**. [CrossRef]
63. Granthi, P.K.; Bansode, S. Android security: A survey of security issues and defenses. *Int. Res. J. Eng. Technol.* **2017**, *4*, 541–549.
64. Gupta, P.; Yadav, K.; Gupta, B.B.; Alazab, M.; Gadekallu, T.R. A Novel Data Poisoning Attack in Federated Learning based on Inverted Loss Function. *Comput. Secur.* **2023**, 103270. [CrossRef]
65. Ahmed, A.; Javed, A.R.; Jalil, Z.; Srivastava, G.; Gadekallu, T.R. Privacy of web browsers: A challenge in digital forensics. In Proceedings of the Genetic and Evolutionary Computing: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computing, Jilin, China, 21–23 October 2021; Springer: Berlin/Heidelberg, Germany, 2022; pp. 493–504.
66. Enck, W.; Octeau, D.; McDaniel, P.D.; Chaudhuri, S. A study of android application security. *Proc. USENIX Secur. Symp.* **2011**, *2*.
67. Ardito, L.; Coppola, R.; Leonardi, S.; Morisio, M.; Buy, U. Automated test selection for Android apps based on APK and activity classification. *IEEE Access* **2020**, *8*, 187648–187670. [CrossRef]
68. Almomani, I.; Khayer, A. Android applications scanning: The guide. In Proceedings of the 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3–4 April 2019; pp. 1–5.
69. Lee, B.s. Changes in the Android App Support Model. In Proceedings of the Korean Institute of Information and Commucation Sciences Conference, Pyeongchang, Republic of Korea, 9–25 Febuary 2019; The Korea Institute of Information and Commucation Engineering: Seoul, Republic of Korea , 2019; pp. 201–203.
70. Roy, D.B.; Fritzmann, T.; Sigl, G. Efficient hardware/software co-design for post-quantum crypto algorithm SIKE on ARM and RISC-V based microcontrollers. In Proceedings of the 39th International Conference on Computer-Aided Design, Virtual Event, 2–5 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 1–9.
71. Pilato, C.; Bohm, S.; Brocheton, F.; Castrillon, J.; Cevasco, R.; Cima, V.; Cmar, R.; Diamantopoulos, D.; Ferrandi, F.; Martinovic, J.; et al. EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms. In Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Virtual, 1–5 February 2021; pp. 1320–1325.
72. Cherif, Z.; Danger, J.L.; Lozac'h, F.; Mathieu, Y.; Bossuet, L. Evaluation of Delay PUFs on CMOS 65 nm Technology: ASIC vs. FPGA. In Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, 23–24 June 2013; pp. 1–8.
73. Pulte, C.; Pichon-Pharabod, J.; Kang, J.; Lee, S.H.; Hur, C.K. Promising-ARM/RISC-V: A simpler and faster operational concurrency model. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Phoenix, AZ, USA, 22–26 June 2019; pp. 1–15.
74. Joshi, J.; Parekh, C. Android smartphone vulnerabilities: A survey. In Proceedings of the 2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA), Greater Noida, India, 29–30 April 2016; pp. 1–5.
75. Asif, S.; Ambreen, M.; Muhammad, Z.; ur Rahman, H.; Iqbal, S. Cloud Computing in Healthcare-Investigation of Threats, Vulnerabilities, Future Challenges and Counter Measure. *LC Int. J. STEM* **2022**, *3*, 63–74.
76. Margossian, H.; Sayed, A.R.J.; Fawaz, W.; Nakad, Z. Partial grid false data injection attacks against state estimation. *Int. J. Electr. Power Energy Syst.* **2019**, *110*, 623–629. [CrossRef]
77. Gandhewar, N.; Sheikh, R. Google Android: An emerging software platform for mobile devices. *Int. J. Comput. Sci. Eng.* **2010**, *1*, 12–17.

78. Rashidi, B.; Fung, C.J. A Survey of Android Security Threats and Defenses. *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl.* **2015**, *6*, 3–35.
79. Shahid, J.; Muhammad, Z.; Iqbal, Z.; Khan, M.S.; Amer, Y.; Si, W. SAT: Integrated Multi-agent Blackbox Security Assessment Tool using Machine Learning. In Proceedings of the 2022 2nd International Conference on Artificial Intelligence (ICAI), Islamabad, Pakistan, 30–31 March 2022; pp. 105–111.
80. Elsersy, W.F.; Feizollah, A.; Anuar, N.B. The rise of obfuscated Android malware and impacts on detection methods. *PeerJ Comput. Sci.* **2022**, *8*, e907. [CrossRef] [PubMed]
81. Rathod, J.; Bhatti, D. Towards a Static and Dynamic Features-Based Framework for Android Vulnerabilities Detection. In Proceedings of the International Joint Conference on Advances in Computational Intelligence, Valletta, Malta, 24–26 October 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 153–166.
82. Selvaganapathy, S.; Sadasivam, S.; Ravi, V. A review on android malware: Attacks, countermeasures and challenges ahead. *J. Cyber Secur. Mobil.* **2021**, 177–230. [CrossRef]
83. Shao, Y.; Lu, Y.; Wei, D.; Fang, J.; Qin, F.; Chen, B. Malicious Code Classification Method Based on Deep Residual Network and Hybrid Attention Mechanism for Edge Security. *Wirel. Commun. Mob. Comput.* **2022**, *2022*. [CrossRef]
84. Moses, A.; Morris, S. Analysis of Mobile Malware: A Systematic Review of Evolution and Infection Strategies. *J. Inf. Secur. Cybercrimes Res.* **2021**, *4*, 103–131.
85. Guerra-Manzanares, A.; Luckner, M.; Bahsi, H. Android Malware Concept Drift using System Calls: Detection, Characterization and Challenges. *Expert Syst. Appl.* **2022**, *206* , 117200. [CrossRef]
86. Bhat, P.; Dutta, K. A survey on various threats and current state of security in android platform. *ACM Comput. Surv.* **2019**, *52*, 1–35. [CrossRef]
87. Gao, J.; Li, L.; Kong, P.; Bissyandé, T.F.; Klein, J. Should you consider adware as malware in your study? In Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), Hangzhou, China, 24–27 February 2019; pp. 604–608.
88. Keyes, D.S.; Li, B.; Kaur, G.; Lashkari, A.H.; Gagnon, F.; Massicotte, F. EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In Proceedings of the 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), Hamilton, ON, Canada, 18–19 May 2021; pp. 1–12.
89. Rehman, F.; Muhammad, Z.; Asif, S.; Rahman, H. The next generation of cloud security through hypervisor-based virtual machine introspection. In Proceedings of the 2023 3rd International Conference on Artificial Intelligence (ICAI), Islamabad, Pakistan, 22 February 2023; pp. 116–121.
90. Pham, A.; Dacosta, I.; Losiouk, E.; Stephan, J.; Huguenin, K.; Hubaux, J.P. Hidemyapp: Hiding the presence of sensitive apps on android. In Proceedings of the 28th USENIX Security Symposium (USENIX Security), Berkeley, CA, USA, 14–16 August 2019, p. 18.
91. Alsoghyer, S.; Almomani, I. Ransomware detection system for Android applications. *Electronics* **2019**, *8*, 868. [CrossRef]
92. Mi, X. *Characterizing Emerging Cybersecurity Threats: An Ecosystem Approach*; Journal Of Indiana University: Bloomington, IN, USA, 2020.
93. Bagui, S.; Brock, H. Machine Learning for Android Scareware Detection. *J. Inf. Technol. Res.* **2022**, *15*, 1–15. [CrossRef]
94. Pierazzi, F.; Mezzour, G.; Han, Q.; Colajanni, M.; Subrahmanian, V. A data-driven characterization of modern Android spyware. *ACM Trans. Manag. Inf. Syst.* **2020**, *11*, 1–38. [CrossRef]
95. Ali, M.; Ali, H.; Anwar, Z. Enhancing Stealthiness & Efficiency of Android Trojans and Defense Possibilities (EnSEAD)-Android's Malware Attack, Stealthiness and Defense: An Improvement. In Proceedings of the 2011 Frontiers of Information Technology, Islamabad, Pakistan, 19–21 December 2011; pp. 148–153.
96. Chen, P.; Desmet, L.; Huygens, C. A study on advanced persistent threats. In Proceedings of the IFIP International Conference on Communications and Multimedia Security, Aveiro, Portugal, 25–26 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 63–72.
97. Kaster, S.D.; Ensign, P.C. Privatized espionage: NSO Group Technologies and its Pegasus spyware. *Thunderbird Int. Bus. Rev.* **2022**. [CrossRef]
98. Tankard, C. Advanced persistent threats and how to monitor and deter them. *Netw. Secur.* **2011**, *2011*, 16–19. [CrossRef]
99. Patil, M.R.; Mulimani, C. Pegasus: Transforming Phone Into A Spy. *Think India J.* **2019**, *22*, 7883–7890.
100. Lee, H.W.; Lee, J. Mobile Forged App Identification System with Centralized Signature Self-verification Method. In Proceedings of the Sixth International Conference on Green and Human Information Technology: ICGHIT 2018, Chiang Mai, Thailand, 31 January–2 Febuary 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 176–182.
101. Pingle, A.; Piplai, A.; Mittal, S.; Joshi, A.; Holt, J.; Zak, R. Relext: Relation extraction using deep learning approaches for cybersecurity knowledge graph improvement. In Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Vancouver, BC, Canada, 27–30 August 2019; pp. 879–886.
102. Ansar, S.A.; Piplai, A.; Mittal, S.; Joshi, A.; Holt, J.; Zak, R. A Critical Analysis of Fraud Cases on the Internet. *Turk. J. Comput. Math. Educ.* **2021**, *12*, 2164–2186.
103. Ichioka, S.; Pouget, E.; Mimura, T.; Nakajima, J.; Yamauchi, T. Accessibility service utilization rates in android applications shared on twitter. In Proceedings of the Information Security Applications: 21st International Conference, WISA 2020, Jeju Island, Republic of Korea, 26–28 August 2020; Revised Selected Papers 21; Springer: Berlin/Heidelberg, Germany, 2020; pp. 101–111.

104. Dhalaria, M.; Gandotra, E. Android malware detection techniques: A literature review. *Recent Patents Eng.* **2021**, *15*, 225–245. [CrossRef]
105. Stevanoski, G.; Kacurova, M.; Bogatinov, D. Rootkits-cyber security challenges and mechanisms for protection. *ETIMA* **2021**, *1*, 174–181.
106. Ramamurthy, M. Fraudster Mobile Apps Detector in Google Playstore. *J. Comput. Theor. Nanosci.* **2020**, *17*, 1752–1757. [CrossRef]
107. Aritonang, J.; Rokhim, R. Big Data Analysis of Paid and Free Applications in Google Playstore and Apple App Store to Know Application Characteristics and Monetization Opportunities for New Startup in Indonesia. In Proceedings of the The International Conference on Business and Management Research (ICBMR 2020), Online, 21–22 October 2020; Atlantis Press: Noord-Holland, The Netherlands , 2020; pp. 205–210.
108. Mirza, S.; Abbas, H.; Shahid, W.B.; Shafqat, N.; Fugini, M.; Iqbal, Z.; Muhammad, Z. A Malware Evasion Technique for Auditing Android Anti-Malware Solutions. In Proceedings of the 2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 23–25 June 2021; pp. 125–130.
109. Glanz, L.; Amann, S.; Eichberg, M.; Reif, M.; Hermann, B.; Lerch, J.; Mezini, M. CodeMatch: Obfuscation will not conceal your repackaged app. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 638–648.
110. Montano, I.H.; de la Torre Díez, I.; López-Izquierdo, R.; Villamor, A.R.J.C.; Martín-Rodríguez, F. Mobile triage applications: A systematic review in the literature and play store. *J. Med Syst.* **2021**, *45*, 1–11. [CrossRef] [PubMed]
111. Cao, M.; Ahmed, K.; Rubin, J. Rotten apples spoil the bunch: An anatomy of Google Play malware. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 21–29 May 2022; pp. 1919–1931.
112. Meacham, M.C.; Vogel, E.A.; Thrul, J. Vaping-related mobile apps available in the Google Play Store after the Apple ban: Content review. *J. Med Internet Res.* **2020**, *22*, e20009. [CrossRef] [PubMed]
113. D'Angelo, G.; Palmieri, F.; Robustelli, A.; Castiglione, A. Effective classification of android malware families through dynamic features and neural networks. *Connect. Sci.* **2021**, *33*, 786–801. [CrossRef]
114. Alazab, M.; Alazab, M.; Shalaginov, A.; Mesleh, A.; Awajan, A. Intelligent mobile malware detection using permission requests and API calls. *Future Gener. Comput. Syst.* **2020**, *107*, 509–521. [CrossRef]
115. Cai, L.; Machiraju, S.; Chen, H. Defending against sensor-sniffing attacks on mobile phones. In Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds, New York, NY, USA, 17 August 2009; pp. 31–36.
116. Sikder, A.K.; Petracca, G.; Aksu, H.; Jaeger, T.; Uluagac, A.S. A survey on sensor-based threats and attacks to smart devices and applications. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1125–1159. [CrossRef]
117. Hubbard, J.; Weimer, K.; Chen, Y. A study of SSL proxy attacks on Android and iOS mobile applications. In Proceedings of the 2014 IEEE 11th Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2014; pp. 86–91.
118. Vidas, T.; Votipka, D.; Christin, N. All your droid are belong to us: A survey of current android attacks. In Proceedings of the 5th USENIX Workshop on Offensive Technologies (WOOT 11), San Francisco, CA, USA, 8 August 2011.
119. Sihombing, P.; Siregar, Y.; Tarigan, J.; Jaya, I.; Turnip, A. Development of building security integration system using sensors, microcontroller and GPS (Global Positioning System) based android smartphone. In *Proceedings of the Journal of Physics: Conference Series*; IOP Publishing: Bristol, UK , 2018; Volume 978, p. 012105.
120. Alrawais, A. Security Issues in Near Field Communications (NFC). *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*. [CrossRef]
121. Tu, Y.J.; Piramuthu, S. On addressing RFID/NFC-based relay attacks: An overview. *Decis. Support Syst.* **2020**, *129*, 113194. [CrossRef]
122. Singh, M.M.; Adzman, K.; Hassan, R. Near Field Communication (NFC) technology security vulnerabilities and countermeasures. *Int. J. Eng. Technol.* **2018**, *7*, 298–305.
123. Shahid, J.; Muhammad, Z.; Iqbal, Z.; Almadhor, A.S.; Javed, A.R. Cellular automata trust-based energy drainage attack detection and prevention in wireless sensor networks. *Comput. Commun.* **2022**. [CrossRef]
124. Senthil Mahesh, P.; Muthumanickam, K. A Security Scheme for Discovering Battery Draining Attacks in Android Smartphone. In *Proceedings of the ICDSMLA 2019*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 1908–1915.
125. Prakash, J.; Sankaran, S.; Jithish, J. Attack Detection based on Statistical Analysis of Smartphone Resource Utilization. In Proceedings of the 2019 IEEE 16th India Council International Conference (INDICON), Rajkot, India, 13–15 December 2019; pp. 1–4.
126. Bala, N.; Ahmar, A.; Li, W.; Tovar, F.; Battu, A.; Bambarkar, P. DroidEnemy: Battling adversarial example attacks for Android malware detection. *Digit. Commun. Netw.* **2021**. [CrossRef]
127. Halawi, B.; Mourad, A.; Otrok, H.; Damiani, E. Few are as good as many: An ontology-based tweet spam detection approach. *IEEE Access* **2018**, *6*, 63890–63904. [CrossRef]
128. Kherraf, N.; Sharafeddine, S.; Assi, C.M.; Ghrayeb, A. Latency and reliability-aware workload assignment in IoT networks with mobile edge clouds. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1435–1449. [CrossRef]
129. Giri, A. A Study on Efficient Battery Management System Providing Features to Resolve Damage occurring in Mobile Phones.
130. Mwinuka, L.J.; Agghey, A.Z.; Kaijage, S.F.; Ndibwile, J.D. FakeAP Detector: An Android-Based Client-Side Application for Detecting Wi-Fi Hotspot Spoofing. *IEEE Access* **2022**, *10*, 13611–13623. [CrossRef]

131. Vanhoef, M. Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation. In Proceedings of the 30th USENIX Security Symposium, Virtual Event, 11–13 August 2021; USENIX Association: Berkeley, CA, USA, 2021.
132. Schrötter, M.; Scheffler, T.; Schnor, B. Evaluation of Intrusion Detection Systems in IPv6 Networks. In Proceedings of the ICETE (2), Prague, Czech Republic, 26–28 July 2019; pp. 408–416.
133. Khazaaleh, S.; Korres, G.; Eid, M.; Rasras, M.; Daqaq, M.F. Vulnerability of MEMS gyroscopes to targeted acoustic attacks. *IEEE Access* **2019**, *7*, 89534–89543. [CrossRef]
134. Guri, M. GAIROSCOPE: Leaking Data from Air-Gapped Computers to Nearby Smartphones using Speakers-to-Gyro Communication. In Proceedings of the 2021 18th International Conference on Privacy, Security and Trust (PST), Auckland, New Zealand, 13–15 December 2021; pp. 1–10.
135. Lin, J.; Seibel, J. Motion-based side-channel attack on mobile keystrokes, 2019.
136. Jaafar, R.H.; Saab, S.S. A neural network approach for indoor fingerprinting-based localization. In Proceedings of the 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 8–10 November 2018; pp. 537–542.
137. Wang, X.; Chen, Y.; Yang, R.; Shi, S.; Lau, W.C. Fingerprint-jacking: Practical fingerprint authorization hijacking in Android apps. *Blackhat Eur. Tech. Rep. Blackhat* **2020**, *2020*.
138. Chugh, T.; Jain, A.K. Fingerprint presentation attack detection: Generalization and efficiency. In Proceedings of the 2019 International Conference on Biometrics (ICB), Crete, Greece, 4–7 June 2019; pp. 1–8.
139. Zhang, R.; Chen, X.; Wen, S.; Zheng, J. Who activated my voice assistant? A stealthy attack on android phones without users' awareness. In Proceedings of the International Conference on Machine Learning for Cyber Security, Xi'an, China, 19–21 September 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 378–396.
140. Zhang, G.; Yan, C.; Ji, X.; Zhang, T.; Zhang, T.; Xu, W. Dolphinattack: Inaudible voice commands. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, Dallas, TX, USA, 30 October–3 November 2017; pp. 103–117.
141. Costa-Pazo, A.; Bhattacharjee, S.; Vazquez-Fernandez, E.; Marcel, S. The replay-mobile face presentation-attack database. In Proceedings of the 2016 International Conference of the Biometrics Special Interest Group (BIOSIG), Darmstadt, Germany, 21–23 September 2016; pp. 1–7.
142. Ye, G.; Tang, Z.; Fang, D.; Chen, X.; Wolff, W.; Aviv, A.J.; Wang, Z. A video-based attack for android pattern lock. *ACM Trans. Priv. Secur.* **2018**, *21*, 1–31. [CrossRef]
143. Morales, A.; Fierrez, J.; Galbally, J.; Gomez-Barrero, M. Introduction to iris presentation attack detection. In *Handbook of Biometric Anti-Spoofing*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 135–150.
144. Gupta, M.; Singh, V.; Vatsa, M.; Singh, R. Detecting Iris spoofing attacks. 2020.
145. Aviv, A.J.; Gibson, K.; Mossop, E.; Blaze, M.; Smith, J.M. Smudge attacks on smartphone touch screens. In Proceedings of the 4th USENIX Workshop on Offensive Technologies (WOOT 10), Berkeley, CA, USA, 9 August 2010.
146. Shahzad, M.; Liu, A.X.; Samuel, A. Behavior based human authentication on touch screen devices using gestures and signatures. *IEEE Trans. Mob. Comput.* **2016**, *16*, 2726–2741. [CrossRef]
147. Shahzad, M.; Liu, A.X.; Samuel, A. Secure unlocking of mobile touch screen devices by simple gestures: You can see it but you can not do it. In Proceedings of the 19th Annual International Conference on Mobile Computing & Networking, New York, NY, USA, 30 September–4 October 2013; pp. 39–50.
148. Imtiaz, S.I.; Khan, L.A.; Almadhor, A.S.; Abbas, S.; Alsubai, S.; Gregus, M.; Jalil, Z. Efficient Approach for Anomaly Detection in Internet of Things Traffic Using Deep Learning. *Wirel. Commun. Mob. Comput.* **2022**. [CrossRef]
149. Song, R.; Song, Y.; Gao, S.; Xiao, B.; Hu, A. I know what you type: Leaking user privacy via novel frequency-based side-channel attacks. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
150. Maiti, A.; Jadliwala, M.; He, J.; Bilogrevic, I. Side-channel inference attacks on mobile keypads using smartwatches. *IEEE Trans. Mob. Comput.* **2018**, *17*, 2180–2194. [CrossRef]
151. Bo, L.; Fengjun, L.; Guanghui, W.; et al. I Know What You Type on Your Phone: Keystroke Inference on Android Device Using Deep Learning. Ph.D. Thesis, University of Kansas, Lawrence, KS, USA , 2019.
152. Kröger, J.L.; Raschke, P.; Bhuiyan, T.R. Privacy implications of accelerometer data: A review of possible inferences. In Proceedings of the 3rd International Conference on Cryptography, Security and Privacy, Kuala Lumpur, Malaysia, 19–21 January 2019; pp. 81–87.
153. Owusu, E.; Han, J.; Das, S.; Perrig, A.; Zhang, J. Accessory: Password inference using accelerometers on smartphones. In Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications, New York, NY, USA, 28–29 February 2012; pp. 1–6.
154. Chen, D.; Zhao, Z.; Qin, X.; Luo, Y.; Cao, M.; Xu, H.; Liu, A. Magleak: A learning-based side-channel attack for password recognition with multiple sensors in IIoT environment. *IEEE Trans. Ind. Inform.* **2020**, *18*, 467–476. [CrossRef]
155. Veerasamy, N. The Threat of Juice Jacking. In Proceedings of the ECCWS 2021 20th European Conference on Cyber Warfare and Security, Online, 24–25 June 2021; Academic Conferences Inter Ltd.: Montreal, QC, Canada , 2021, p. 449.
156. Spolaor, R.; Abudahi, L.; Moonsamy, V.; Conti, M.; Poovendran, R. No free charge theorem: A covert channel via usb charging cable on mobile devices. In Proceedings of the International Conference on Applied Cryptography and Network Security, Kanazawa, Japan, 10–12 July 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 83–102.

157. Kumar, Y. Juice Jacking-The USB Charger Scam. *Available at SSRN 3580209* **2020**. [CrossRef]
158. Goodin, D. Hackers Have Been Exploiting 4 Critical Android Vulnerabilities. *Ars Technica* **2021**.
159. Qiu, P.; Wang, D.; Lyu, Y.; Tian, R.; Wang, C.; Qu, G. Voltjockey: A new dynamic voltage scaling-based fault injection attack on intel sgx. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *40*, 1130–1143. [CrossRef]
160. Gao, J.; Xu, Y.; Jiang, Y.; Liu, Z.; Chang, W.; Jiao, X.; Sun, J. Em-fuzz: Augmented firmware fuzzing via memory checking. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 3420–3432. [CrossRef]
161. Melotti, D.; Rossi-Bellom, M.; Continella, A. Reversing and fuzzing the google titan m chip. In Proceedings of the Reversing and Offensive-Oriented Trends Symposium, Vienna, Austria, 28–29 November 2021; pp. 1–10.
162. Cheng, J.; Liu, W.; Sun, N.; Peng, Z.; Sun, C.; Wang, C.; Bi, Y.; Wen, Y.; Zhang, H.; Zhang, P.; et al. A machine learning low-dropout regulator-assisted differential power analysis attack countermeasure with voltage scaling. *Int. J. Circuit Theory Appl.* **2023**. [CrossRef]
163. Aminuddin, A. Android Assets Protection Using RSA and AES Cryptography to Prevent App Piracy. In Proceedings of the 2020 3rd International Conference on Information and Communications Technology (ICOIACT), Yogyakarta, Indonesia, 24–25 November 2020; pp. 461–465.
164. Banik, S.; Bogdanov, A.; Isobe, T.; Shibutani, K.; Hiwatari, H.; Akishita, T.; Regazzoni, F. Midori: A block cipher for low energy. In *Proceedings of the Advances in Cryptology–ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, 29 November–3 December 2015*; Part II 21; Springer: Berlin/Heidelberg, Germany, 2015; pp. 411–436.
165. Fahrianto, F.; et al. End-To-End Encryption on the Instant Messaging Application Based Android using AES Cryptography Algorithm to a Text Message. In Proceedings of the 2022 10th International Conference on Cyber and IT Service Management (CITSM), Yogyakarta, Indonesia, 20–21 September 2022; pp. 1–6.
166. Li, H.; Shen, L.; Wang, Y.; Feng, J.; Tan, H.; Li, Z. Risk measurement method of collusion privilege escalation attacks for android apps based on feature weight and behavior determination. *Secur. Commun. Netw.* **2021**, *2021*. [CrossRef]
167. Bhandari, S.; Laxmi, V.; Zemmari, A.; Gaur, M.S. Intersection automata based model for android application collusion. In Proceedings of the 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, Switzerland, 23–25 March 2016; pp 901–908.
168. Bhandari, S.; Jaballah, W.B.; Jain, V.; Laxmi, V.; Zemmari, A.; Gaur, M.S.; Mosbah, M.; Conti, M. Android inter-app communication threats and detection techniques. *Comput. Secur.* **2017**, *70*, 392–421. [CrossRef]
169. Liu, F.; Cai, H.; Wang, G.; Yao, D.; Elish, K.O.; Ryder, B.G. MR-Droid: A scalable and prioritized analysis of inter-app communication risks. In Proceedings of the 2017 IEEE Security and Privacy Workshops (SPW), San Jose, CA, USA, 25 May 2017; pp. 189–198.
170. Elish, K.O.; Cai, H.; Barton, D.; Yao, D.; Ryder, B.G. Identifying mobile inter-app communication risks. *IEEE Trans. Mob. Comput.* **2018**, *19*, 90–102. [CrossRef]
171. Casolare, R.; Di Giacomo, U.; Martinelli, F.; Mercaldo, F.; Santone, A. Android Collusion Detection by means of Audio Signal Analysis with Machine Learning techniques. *Procedia Comput. Sci.* **2021**, *192*, 2340–2346. [CrossRef]
172. Lee, Y.K.; Bang, J.Y.; Safi, G.; Shahbazian, A.; Zhao, Y.; Medvidovic, N. A sealant for inter-app security holes in android. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, 20–28 May 2017; pp. 312–323.
173. Stang, J.; Dmitrienko, A.; Roth, S. RIP StrandHogg: A practical StrandHogg attack detection method on Android. In Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, New York, NY, USA, 28 June–2 July 2021; pp. 216–226.
174. Escobar, F.S.; da Silva, A.S.; Vergara, L.O.C. Nova Vulnerabilidade DO Android. *Semin. Technol. Manag. Educ.* **2020**, *2*.
175. Eliassen, K.O. Strandens topologier. *K&K-Kultur Klasse* **2020**, *48*, 177–208.
176. Sun, P.; Chen, S.; Fan, L.; Gao, P.; Song, F.; Yang, M. VenomAttack: Automated and Adaptive Activity Hijacking in Android.
177. Κασαγιάννης, G. Security Evaluation of Android Keystore. Master's Thesis, University of Piraeus, Pireas, Greece, 2018.
178. Focardi, R.; Palmarini, F.; Squarcina, M.; Steel, G.; Tempesta, M. Mind Your Keys? A Security Evaluation of Java Keystores. In Proceedings of the NDSS, San Diego, CA, USA, 18–21 February 2018.
179. Sabt, M.; Traoré, J. Breaking into the keystore: A practical forgery attack against Android keystore. In *Proceedings of the European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 531–548.
180. Chalhoub, M.; Khazzaka, A.; Sarkis, R.; Sleiman, Z. The role of smartphone game applications in improving laparoscopic skills. *Adv. Med Educ. Pract.* **2018**, 541–547. [CrossRef] [PubMed]
181. Chehab, M.; Mourad, A. Towards a lightweight policy-based privacy enforcing approach for IoT. In Proceedings of the 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 12–14 December 2018; pp. 984–989.
182. Bugiel, S.; Davi, L.; Dmitrienko, A.; Fischer, T.; Sadeghi, A.R.; Shastry, B. Towards Taming Privilege-Escalation Attacks on Android. *Proc. NDSS Citeseer* **2012**, *17*, 19.
183. Costamagna, V.; Zheng, C.; Huang, H. Identifying and evading android sandbox through usage-profile based fingerprints. In Proceedings of the First Workshop on Radical and Experiential Security, New York, NY, USA, 4 June 2018; pp. 17–23.

184. Crosta, P.; Serruys, H.; Watterton, T.; Galluzzo, G.; Lucas, R. Authentication of GNSS orbital and clock parameters at android application layer. In Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019), Miami, FL, USA, 16–20 September 2019; pp. 290–298.
185. Zhang, W.; Su, N.; Niu, S.; Li, H.; Huang, R. A Novel Hotfix Scheme for System Vulnerability Based on the Android Application Layer. *Chin. J. Electron.* **2019**, *28*, 408–415. [CrossRef]
186. Wang, W.; Fida, M.H.; Lian, Z.; Yin, Z.; Pham, Q.V.; Gadekallu, T.R.; Dev, K.; Su, C. Secure-enhanced federated learning for ai-empowered electric vehicle energy prediction. *IEEE Consum. Electron. Mag.* **2021**. [CrossRef]
187. Shen, L.; Li, H.; Wang, H.; Wang, Y. Multifeature-based behavior of privilege escalation attack detection method for android applications. *Mob. Inf. Syst.* **2020**, *2020*. [CrossRef]
188. Xiang, X.; Zhang, R.; Wen, H.; Gong, X.; Liu, B. Ghost in the Binder: Binder Transaction Redirection Attacks in Android System Services. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 15–19 November 2021; pp. 1581–1597.
189. Ma, H.; Li, S.; Gao, D.; Wu, D.; Jia, Q.; Jia, C. Active warden attack: On the (in) effectiveness of Android app repackage-proofing. *IEEE Trans. Dependable Secur. Comput.* **2021**. [CrossRef]
190. Sun, X.; Han, J.; Dai, H.; Li, Q. An active android application repacking detection approach. In Proceedings of the 2018 10th International Conference on Communication Software and Networks (ICCSN), Chengdu, China, 6–9 July 2018 ; pp. 493–496.
191. Shaik, A.; Borgaonkar, R.; Park, S.; Seifert, J.P. New vulnerabilities in 4G and 5G cellular access network protocols: Exposing device capabilities. In Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks, Miami, FL, USA, 15–17 May 2019; pp. 221–231.
192. Zeqiri, R.; Idrizi, F.; Halimi, H. Comparison of Algorithms and Technologies 2G, 3G, 4G and 5G. In Proceedings of the 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 11–13 October 2019; pp. 1–4.
193. Fang, K.; Yan, G. Paging storm attacks against 4G/LTE networks from regional Android botnets: Rationale, practicality, and implications. In Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks, New York, NY, USA, 8–10 July 2020; pp. 295–305.
194. Qasmi, W.N.A. Cellular Networks under Signalling Attacks. Ph.D. Thesis, Lahore University of Management Sciences, Punjab, Pakistan , 2019.
195. Shaikhanov, Z.; Hassan, F.; Guerboukha, H.; Mittleman, D.; Knightly, E. Metasurface-in-the-middle attack: From theory to experiment. In Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks, New York, NY, USA, 16–19 May 2022; pp. 257–267.
196. Ankita, A.; Rani, S. Machine Learning and Deep Learning for Malware and Ransomware Attacks in 6G Network. In Proceedings of the 2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT), Sonepat, India, 3 July 2021; pp. 39–44.
197. Mone, G. The quantum threat. *Commun. ACM* **2020**, *63*, 12–14. [CrossRef]
198. Niraula, T.; Pokharel, A.; Phuyal, A.; Palikhel, P.; Pokharel, M. Quantum computers' threat on current cryptographic measures and possible solutions. *Int. J. Wirel. Microw. Technol.* **2022**, *12*, 10–20. [CrossRef]
199. Kaddoura, S.; Haraty, R.A.; Al Kontar, K.; Alfandi, O. A parallelized database damage assessment approach after cyberattack for healthcare systems. *Future Internet* **2021**, *13*, 90. [CrossRef]
200. Abbas, N.; Nasser, Y.; Shehab, M.; Sharafeddine, S. Attack-specific feature selection for anomaly detection in software-defined networks. In Proceedings of the 2021 3rd IEEE Middle East and North Africa COMMunications Conference (MENACOMM), Agadir, Morocco, 3–5 December 2021; pp. 142–146.
201. Borkar, T.; Heide, F.; Karam, L. Defending against universal attacks through selective feature regeneration. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Seattle, WA, USA, 13–19 June 2020; pp. 709–719.
202. Xu, G.; Mao, J.; Sakk, E.; Wang, S.P. An Overview of Quantum-Safe Approaches: Quantum Key Distribution and Post-Quantum Cryptography. In Proceedings of the 2023 57th Annual Conference on Information Sciences and Systems (CISS), Hopkins, MN, USA, 23–24 March 2023; pp. 1–6.
203. Joseph, D.; Misoczki, R.; Manzano, M.; Tricot, J.; Pinuaga, F.D.; Lacombe, O.; Leichenauer, S.; Hidary, J.; Venables, P.; Hansen, R. Transitioning organizations to post-quantum cryptography. *Nature* **2022**, *605*, 237–243. [CrossRef]
204. Sharma, R.M.; Agrawal, C.; Kumar, V.; Mulatu, A.N. CFSBFDroid: Android Malware Detection Using CFS+ Best First Search-Based Feature Selection. *Mob. Inf. Syst.* **2022**, *2022*. [CrossRef]
205. Ou, F.; Xu, J. S3Feature: A static sensitive subgraph-based feature for android malware detection. *Comput. Secur.* **2022**, *112*, 102513. [CrossRef]
206. Sasidharan, S.K.; Thomas, C. ProDroid—An Android malware detection framework based on profile hidden Markov model. *Pervasive Mob. Comput.* **2021**, *72*, 101336. [CrossRef]
207. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Comput. Secur.* **2020**, *89*, 101663. [CrossRef]
208. Lee, W.Y.; Saxe, J.; Harang, R. SeqDroid: Obfuscated Android malware detection using stacked convolutional and recurrent neural networks. In *Deep Learning Applications for Cyber Security*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 197–210.

209. Xu, K.; Li, Y.; Deng, R.; Chen, K.; Xu, J. Droidevolver: Self-evolving android malware detection system. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, 17–19 June 2019; pp. 47–62.
210. Riad, K.; Ke, L. RoughDroid: Operative scheme for functional android malware detection. *Secur. Commun. Netw.* **2018**, *2018*. [CrossRef]
211. Karbab, E.B.; Debbabi, M.; Derhab, A.; Mouheb, D. MalDozer: Automatic framework for android malware detection using deep learning. *Digit. Investig.* **2018**, *24*, S48–S59. [CrossRef]
212. Hou, S.; Ye, Y.; Song, Y.; Abdulhayoglu, M. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, New York, NY, USA, 13–17 August 2017; pp. 1507–1515.
213. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. DynaLog: An automated dynamic analysis framework for characterizing android applications. In Proceedings of the 2016 International Conference on Cyber Security Furthermore, Protection of Digital Services (Cyber Security), London, UK, 13–14 June 2016; pp. 1–8.
214. Xu, K.; Li, Y.; Deng, R.H. Iccdetector: Icc-based malware detection on android. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 1252–1264. [CrossRef]
215. Talha, K.A.; Alper, D.I.; Aydin, C. APK Auditor: Permission-based Android malware detection system. *Digit. Investig.* **2015**, *13*, 1–14. [CrossRef]
216. Arzt, S.; Rasthofer, S.; Fritz, C.; Bodden, E.; Bartel, A.; Klein, J.; Le Traon, Y.; Octeau, D.; McDaniel, P. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM Sigplan Not.* **2014**, *49*, 259–269. [CrossRef]
217. Zhang, Y.; Yang, M.; Xu, B.; Yang, Z.; Gu, G.; Ning, P.; Wang, X.S.; Zang, B. Vetting undesirable behaviors in android apps with permission use analysis. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 611–622.
218. Wu, D.J.; Mao, C.H.; Wei, T.E.; Lee, H.M.; Wu, K.P. Droidmat: Android malware detection through manifest and api calls tracing. In Proceedings of the 2012 Seventh Asia joint conference on information security, Tokyo, Japan, 9–10 August 2012; pp. 62–69.
219. Iland, D.; Pucher, A.; Schauble, T. Detecting android malware on network level. *Univ. Calif. Santa Barbar.* **2011**, *12*.
220. Shabtai, A.; Fledel, Y.; Kanonov, U.; Elovici, Y.; Dolev, S. Google android: A state-of-the-art review of security mechanisms. *arXiv* **2009**, arXiv:0912.5101.
221. Chess, B.; McGraw, G. Static analysis for security. *IEEE Secur. Priv.* **2004**, *2*, 76–79. [CrossRef]
222. Landi, W. Undecidability of static analysis. *ACM Lett. Program. Lang. Syst.* **1992**, *1*, 323–337. [CrossRef]
223. Li, L.; Bissyandé, T.F.; Papadakis, M.; Rasthofer, S.; Bartel, A.; Octeau, D.; Klein, J.; Traon, L. Static analysis of android apps: A systematic literature review. *Inf. Softw. Technol.* **2017**, *88*, 67–95. [CrossRef]
224. Ball, T. The concept of dynamic analysis. In Proceedings of the Software Engineering—ESEC/FSE'99, Toulouse, France, 6–10 September 1999; Springer: Berlin/Heidelberg, Germany, 1999; pp. 216–234.
225. Vamvatsikos, D.; Cornell, C.A. Incremental dynamic analysis. *Earthq. Eng. Struct. Dyn.* **2002**, *31*, 491–514. [CrossRef]
226. Wong, M.Y.; Lie, D. Intellidroid: A targeted input generator for the dynamic analysis of android malware. *Proc. NDSS* **2016**, *16*, 21–24.
227. Cintas-Canto, A.; Mozaffari-Kermani, M.; Azarderakhsh, R.; Gaj, K. CRC-oriented error detection architectures of post-quantum cryptography niederreiter key generator on FPGA. In Proceedings of the 2022 IEEE Nordic Circuits and Systems Conference (NorCAS), Oslo, Norway, 25–26 October 2022; pp. 1–7.
228. Mozaffari-Kermani, M.; Azarderakhsh, R.; Aghaie, A. Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC. *ACM Trans. Embed. Comput. Syst.* **2016**, *16*, 1–19. [CrossRef]
229. Canto, A.C.; Kermani, M.M.; Azarderakhsh, R. Reliable constructions for the key generator of code-based post-quantum cryptosystems on FPGA. *ACM J. Emerg. Technol. Comput. Syst.* **2022**, *19*, 1–20. [CrossRef]
230. Anastasova, M.; Azarderakhsh, R.; Kermani, M.M.; Beshaj, L. Time-Efficient Finite Field Microarchitecture Design for Curve448 and Ed448 on Cortex-M4. In Proceedings of the Information Security and Cryptology–ICISC 2022: 25th International Conference, ICISC 2022, Seoul, Republic of Korea, 30 November–2 December 2022; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2023; pp. 292–314.
231. Anastasova, M.; Bisheh-Niasar, M.; Seo, H.; Azarderakhsh, R.; Kermani, M.M. Efficient and Side-Channel Resistant Design of High-Security Ed448 on ARM Cortex-M4. In Proceedings of the 2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 27–30 June 2022; pp. 93–96.
232. Bisheh Niasar, M.; Azarderakhsh, R.; Kermani, M.M. Efficient hardware implementations for elliptic curve cryptography over Curve448. In Proceedings of the Progress in Cryptology–INDOCRYPT 2020: 21st International Conference on Cryptology in India, Bangalore, India, 13–16 December 2020; Proceedings 21; Springer: Berlin/Heidelberg, Germany, 2020; pp. 228–247.
233. Bruno, G.; Batina, L.; Bosma, W. *Crypto Security Optimizations*; Radboud University Nijmegen: Nijmegen, The Netherlands , 2021.
234. Anastasova, M.; Bisheh-Niasar, M.; Azarderakhsh, R.; Kermani, M.M. Compressed SIKE Round 3 on ARM Cortex-M4. In Proceedings of the Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, 6–9 September 2021; Proceedings, Part II 17; Springer: Berlin/Heidelberg, Germany, 2021; pp. 441–457.
235. Anastasova, M.; Azarderakhsh, R.; Kermani, M.M. Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. *IEEE Trans. Circuits Syst. Regul. Pap.* **2021**, *68*, 4129–4141. [CrossRef]

236. Elkhatib, R.; Koziel, B.; Azarderakhsh, R. Faster Isogenies for Post-quantum Cryptography: SIKE. In Proceedings of the Topics in Cryptology–CT-RSA 2022: Cryptographers' Track at the RSA Conference 2022, Virtual Event, 1–2 March 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 49–72.
237. Sanal, P.; Karagoz, E.; Seo, H.; Azarderakhsh, R.; Mozaffari-Kermani, M. Kyber on ARM64: Compact implementations of Kyber on 64-bit ARM Cortex-A processors. In Proceedings of the Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, 6–9 September 2021; Part II; Springer: Berlin/Heidelberg, Germany, 2021; pp. 424–440.
238. Kwon, H.; Kim, H.; Sim, M.; Lee, W.K.; Seo, H. Look-up the Rainbow: Efficient Table-based Parallel Implementation of Rainbow Signature on 64-bit ARMv8 Processors. *Cryptol. Eprint Arch.* **2021**.
239. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari-Kermani, M. Cryptographic accelerators for digital signature based on Ed25519. *IEEE Trans. Very Large Scale Integr. (Vlsi) Syst.* **2021**, *29*, 1297–1305. [CrossRef]
240. Hoang, T.T.; Duran, C.; Serrano, R.; Sarmiento, M.; Nguyen, K.D.; Tsukamoto, A.; Suzaki, K.; Pham, C.K. Trusted execution environment hardware by isolated heterogeneous architecture for key scheduling. *IEEE Access* **2022**, *10*, 46014–46027. [CrossRef]
241. Malina, L.; Cibik, P.; Jedlicka, P.; Smekal, D.; Ricci, S.; Hrabovsky, J. Hardware-based Cryptographic Accelerator for Post Quantum Era. In Proceedings of the 2021 13th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Brno, Czech Republic, 25–27 October 2021; pp. 149–155.
242. Bauer, S.; Rass, S.; Schartner, P. Generic parity-based concurrent error detection for lightweight ARX ciphers. *IEEE Access* **2020**, *8*, 142016–142025. [CrossRef]
243. Mozaffari-Kermani, M.; Azarderakhsh, R.; Aghaie, A. Reliable and error detection architectures of Pomaranch for false-alarm-sensitive cryptographic applications. *IEEE Trans. Very Large Scale Integr. Syst.* **2015**, *23*, 2804–2812. [CrossRef]
244. Gowri, B.; Dhanyasri, J. An FPGA Implementation of Fault Diagnosis Architecture of S-Box For Cryptographic Application. *Int. J. Multidiscip. Res.* **2018**, *3*, 2395–7964 .
245. Mozaffari-Kermani, M.; Reyhani-Masoleh, A. Reliable hardware architectures for the third-round SHA-3 finalist Grostl benchmarked on FPGA platform. In Proceedings of the 2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Washington, DC, USA, 3–5 October 2011; pp. 325–331.
246. Fischer, W.; Reuter, C.A. Differential fault analysis on Grøstl. In Proceedings of the 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, 9 September 2012; pp. 44–54.
247. Aghaie, A.; Kermani, M.M.; Azarderakhsh, R. Fault diagnosis schemes for low-energy block cipher Midori benchmarked on FPGA. *IEEE Trans. Very Large Scale Integr. Syst.* **2016**, *25*, 1528–1536. [CrossRef]
248. Zhang, W.; Bao, Z.; Lin, D.; Rijmen, V.; Yang, B.; Verbauwhede, I. RECTANGLE: A bit-slice lightweight block cipher suitable for multiple platforms. *Cryptol. Eprint Arch.* **2014**. [CrossRef]
249. Wei, L.; Liu, Y.; Cheung, S.C. Taming android fragmentation: Characterizing and detecting compatibility issues for android apps. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, 3–7 September 2016; pp. 226–237.
250. Farhang, S.; Laszka, A.; Grossklags, J. An economic study of the effect of android platform fragmentation on security updates. In Proceedings of the Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, 26 February–2 March 2018; Revised Selected Papers 22; Springer: Berlin/Heidelberg, Germany, 2018; pp. 119–137.
251. Nguyen-Vu, L.; Ahn, J.; Jung, S. Android fragmentation in malware detection. *Comput. Secur.* **2019**, *87*, 101573. [CrossRef]
252. Park, J.H.; Park, Y.B.; Ham, H.K. Fragmentation problem in Android. In Proceedings of the 2013 International Conference on Information Science and Applications (ICISA), Pattaya, Thailand, 24–26 June 2013; pp. 1–2.
253. He, Y.; Yang, X.; Hu, B.; Wang, W. Dynamic privacy leakage analysis of Android third-party libraries. *J. Inf. Secur. Appl.* **2019**, *46*, 259–270. [CrossRef]
254. Zhan, X.; Fan, L.; Chen, S.; We, F.; Liu, T.; Luo, X.; Liu, Y. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, Spain, 22–30 May 2021; pp. 1695–1707.
255. Ma, Z.; Wang, H.; Guo, Y.; Chen, X. Libradar: Fast and accurate detection of third-party libraries in android apps. In Proceedings of the 38th International Conference on Software Engineering Companion, Austin, TX, USA, 14–22 May 2016; pp. 653–656.
256. Zhang, L.; Liu, C.; Xu, Z.; Chen, S.; Fan, L.; Zhao, L.; Wu, J.; Liu, Y. Compatible Remediation on Vulnerabilities from Third-Party Libraries for Java Projects. *arXiv* **2023**, arXiv:2301.08434.
257. Chehab, M.; Mourad, A. Lp-sba-xacml: Lightweight semantics based scheme enabling intelligent behavior-aware privacy for iot. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 161–175. [CrossRef]
258. Liu, X.; Liu, J.; Zhu, S.; Wang, W.; Zhang, X. Privacy risk analysis and mitigation of analytics libraries in the android ecosystem. *IEEE Trans. Mob. Comput.* **2019**, *19*, 1184–1199. [CrossRef]
259. Rehman, A.; Razzak, I.; Xu, G. Federated learning for privacy preservation of healthcare data from smartphone-based side-channel attacks. *IEEE J. Biomed. Health Inform.* **2022**. [CrossRef]
260. AbdulRahman, S.; Tout, H.; Mourad, A.; Talhi, C. FedMCCS: Multicriteria client selection model for optimal IoT federated learning. *IEEE Internet Things J.* **2020**, *8*, 4723–4735. [CrossRef]
261. Rahman, S.A.; Tout, H.; Talhi, C.; Mourad, A. Internet of things intrusion detection: Centralized, on-device, or federated learning? *IEEE Netw.* **2020**, *34*, 310–317. [CrossRef]

262. AbdulRahman, S.; Tout, H.; Ould-Slimane, H.; Mourad, A.; Talhi, C.; Guizani, M. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet Things J.* **2020**, *8*, 5476–5497. [CrossRef]
263. Wahab, O.A.; Mourad, A.; Otrok, H.; Taleb, T. Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 1342–1397. [CrossRef]
264. Qu, Z.; Alam, S.; Chen, Y.; Zhou, X.; Hong, W.; Riley, R. DyDroid: Measuring dynamic code loading and its security implications in Android applications. In Proceedings of the 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Denver, CO, USA, 26–29 June 2017; pp. 415–426.
265. Shaikh, S.; Rupa, C.; Srivastava, G.; Gadekallu, T.R. Botnet Attack Intrusion Detection In IoT Enabled Automated Guided Vehicles. In Proceedings of the 2022 IEEE International Conference on Big Data (Big Data), Osaka, Japan, 17–20 December 2022; pp. 6332–6336.
266. Gookyi, N.; Agyemanh, D.; Kanda, G.; Ryoo, K. NIST Lightweight Cryptography Standardization Process: Classification of Second Round Candidates, Open Challenges, and Recommendations. *J. Inf. Process. Syst.* **2021**, *17*.
267. Altınay, Ö.; Örs, B. Instruction extension of RV32I and GCC back end for Ascon lightweight cryptography algorithm. In Proceedings of the 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Barcelona, Spain, 23–26 August 2021; pp. 1–6.