Smartphone Security and Privacy

Subjects: Computer Science, Information Systems | Computer Science, Artificial Intelligence

Contributor: Zia Muhammad , Zahid Anwar , Abdul Rehman Javed , Bilal Saleem , Sidra Abbas , Thippa Reddy Gadekallu

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.

smartphone security | security and privacy | android issues | malware attacks

1. 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 Vulnerabilities can exist in various forms, such as bugs in the code, outdated software, or misconfigured settings [1][2][3]. 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 [3][4][5].

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 [6][7][8].

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.

2. Cyber Threats

2.1. 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. 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 [9][10].

2.1.1. 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 [11].

2.1.2. 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 [12].

2.1.3. File Infector

File infectors are malware that attach themselves to files, such as APKs, to access an application's data [13][14]. This can allow malware to conduct harmful actions, such as stealing sensitive information or encrypting files.

2.1.4. 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 [15]. Examples include applications that display unwanted advertisements or track user activity.

2.1.5. 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 [16].

2.1.6. 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 [17].

2.1.7. 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 [18].

2.1.8. 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 [19].

2.1.9. 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 [20].

2.2. 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 [21]. 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 [22]. 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 [23]. Here are some of the APTs that were detected in recent years.

2.2.1. 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 [24]. Pegasus exploits several vulnerabilities in the operating system and various apps, such as WhatsApp, commonly used by targeted individuals.

2.2.2. Gooligan

Gooligan is an APT typically spread through malicious apps and can take control of infected devices [25]. 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.

2.2.3. 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 [26].

2.2.4. Hornbill

Hornbill is an Android APT malware capable of stealing sensitive information from infected devices, including text messages, call logs, and contact lists [27]. 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.

2.2.5. 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 [28]. 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.

2.2.6. Skygofree

Skygofree is an APT capable of recording audio and video, taking screenshots, and stealing sensitive data from infected devices [29]. 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.

2.2.7. Triout

Triout is an APT designed to steal data from infected devices, including call logs, text messages, and location data [30]. 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.

2.2.8. Mosaic Regressor

Mosaic regressor is an APT that targets Android devices. It is typically spread through phishing messages that contain links to malicious websites [31]. 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.

3. 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 [32]. 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 [33][34][35]. There are several means by which attackers can upload and propagate malware through the Google Play Store, the details of which are as follows.

- **Dynamic code loading:** Attackers can use dynamic code loading techniques, such as Java reflection or Android DexClassLoader, to load and execute code at runtime [36]. 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.

- **Incremental malicious updates attack (IMUTA):** Attackers can use incremental updates to gradually add malicious code to an application over time [33]. 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.

- **Code obfuscation:** Obfuscation consists of modifying an app's source code to make it more difficult to understand or analyze [37]. 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 [38].

- **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 [34]. 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.

- **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 [34][39].

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.

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.

- **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.

4. Sensor-Based Attacks

4.1. 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 [40]. 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 [40]. 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 [41][42]. These attacks can have severe consequences, such as privacy violations, location-based scams, or even physical harm in some cases [43]. Moreover, malicious applications with GPS information rights can log individuals with precise location coordinates and track their movements [44].

4.2. 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 [45]. 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 [46]. 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) [47]. These attacks can be carried out using specialized hardware or software tools and compromise personal and financial information security.

4.3. 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 [48]. In such attacks, an attacker utilizes maximum resources to drain the device's power [49][50]. 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 [51][52][53][54][55].

4.4. 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 [56]. 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" [57]. 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 [58].

4.5. 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 [59]. 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) [60], 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 [61].

4.6. 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 [62]. 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.

5. Side-Channel Attacks

5.1. 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 [63]. 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 [64][65][66].

5.2. 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 [61][67][68]. An attacker demonstrated an attack named AlphaLogger, an Android-based application that can induce letter keys, typed on a soft keyboard [69]. 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 [70].

5.3. 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 [71]. 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 [72][73].

5.4. Juice Jacking Attacks

Juice jacking attacks typically involve malicious charging cables, USB ports, and wall adapters designed to look like legitimate charging stations [74]. 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 [75][76].

5.5. 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 [77]. Some of the common techniques for fault injection on Android devices are as follows:

- **Voltage fault injection:** This involves manipulating the power supply of the device to cause glitches in the CPU or memory. For example, VoltJockey3 [78] is an attack that uses voltage scaling to inject faults in the ARM TrustZone and compromise the secure world execution.

- **Electromagnetic fault injection:** This involves applying electromagnetic pulses to the device to induce transient faults in the circuits. For example, EM-Fuzz [79] 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 [80] 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 [81] 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.

5.6. 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 [82]. 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 [80] 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 [83] can use cryptographic techniques to protect encryption keys from power analysis attacks.

5.7. 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) [84]. 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 [85].

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.

6. Intrinsic Cyberattacks

6.1. 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 [86]. This is performed by splitting a malicious functionality within multiple applications and exfiltrates data by exploiting the inter-app communication feature [87]. 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.

6.2. 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 [88][89][90]. 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 [91][92].

6.3. StrandHogg Attack

StrandHogg is an Android vulnerability (critical severity: CVE-2020-0096) through which a malicious application is installed on the operating system [93]. 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 [94][95]. 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 [96].

6.4. 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 [97][98]. 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 [99]. 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 [99]. 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 [98][100][101].

6.5. 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 [102]. 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 [103]. 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.

6.6. 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 [104]. 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) [105]. Due to these open ports, the device becomes vulnerable to cyberattacks such as security misconfiguration, SQL injection, and cross-site scripting [106]. The attacker can exploit any vulnerability present to deceive the user into controlling the application level [107].

6.7. 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 [108]. 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.

6.8. 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 [109]. The goal of an AWA is to evade detection using steganography to hide the malicious functionality within the app [109]. An attacker may use AWA to repackage and distribute an application to users, potentially stealing sensitive information or performing other malicious actions [110].

6.9. 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 [111][112]. 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.

6.10. 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 [112][113]. 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.

6.11. 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 [112]. 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.

6.12. 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 [114]. 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 [115]. This successful demonstration highlights the need for more security enhancements to protect against potential attacks on 6G networks [116].

6.13. 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 [117]. 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 [118]. 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 [119][120][121].

7. Threat Detection and Mitigation

7.1. 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 [122]. 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 [123]. 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 [36]. 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 [124].

- *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.

- *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.

- *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.
- *String analysis:* This technique involves identifying malware by searching for hardcoded 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.
- *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.
- *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.
- *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.
- *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.

7.2. Dynamic Malware Analysis

Dynamic malware analysis on Android is a technique used to analyze the behavior of an Android application while running [125]. This technique executes the malicious APK in a controlled and monitored environment such as an application sandbox, emulators, or virtual machines [126]. 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 [127]. This technique uses various detection methods to identify malicious activity in an application. Some of the common methods used are:

- *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.
- *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.

- *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.
- *API calls:* This involves monitoring the application's application programming interfaces (APIs) to detect attempts to access restricted resources or perform malicious actions.
- *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.
- *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.
- *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.
- *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.

7.3. 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 [128]. 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 [129][130]. Details are outlined as follows.

7.3.1. 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 [131].

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 [132]. 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 [133]. 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.

7.3.2. 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 [131]. 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 [134]. This rigorous testing helps strengthen the overall security of the SIKE implementation on Cortex-M4, ensuring its resilience against potential attacks.

7.3.3. 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 [135]. 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 [136]. 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 [137].

7.3.4. 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 [138]. 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 [139]. 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.

7.3.5. 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 [140]. 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 [141]. 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 [142]. 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.

7.4. 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 [143][144]. 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.

7.4.1. 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 [144].

7.4.2. 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 [145]. 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 [146].

7.4.3. 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. [147] 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.

7.4.4. 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 [148]. 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. [147] 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.

8. Open Issues and Challenges

8.1. 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 [149][150]. 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 [151]. 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 [152]. 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.

8.2. 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 [153]. 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 [154].

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 [155]. 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 [156][157][158].

8.3. 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 [159]. 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 [36][160]. 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.

8.4. 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.

8.5. 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 [161]. 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 [162]. 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.

References

1. 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.
2. 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.

3. 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.
4. 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.
5. Gandhewar, N.; Sheikh, R. Google Android: An emerging software platform for mobile devices. Int. J. Comput. Sci. Eng. 2010, 1, 12–17.
6. 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.
7. 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.
8. 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.
9. Tan, D.J.; Chua, T.W.; Thing, V.L. Securing android: A survey, taxonomy, and challenges. ACM Comput. Surv. 2015, 47, 1–45.
10. Bhat, P.; Dutta, K. A survey on various threats and current state of security in android platform. ACM Comput. Surv. 2019, 52, 1–35.
11. 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.
12. Garg, S.; Baliyan, N. Comparative analysis of Android and iOS from security viewpoint. Comput. Sci. Rev. 2021, 40, 100372.
13. 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.
14. 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.
15. 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.

16. Alsoghyer, S.; Almomani, I. Ransomware detection system for Android applications. Electronics 2019, 8, 868.
17. Mi, X. Characterizing Emerging Cybersecurity Threats: An Ecosystem Approach; Journal Of Indiana University: Bloomington, IN, USA, 2020.
18. Bagui, S.; Brock, H. Machine Learning for Android Scareware Detection. J. Inf. Technol. Res. 2022, 15, 1–15.
19. 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.
20. 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.
21. 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.
22. Kaster, S.D.; Ensign, P.C. Privatized espionage: NSO Group Technologies and its Pegasus spyware. Thunderbird Int. Bus. Rev. 2022.
23. Tankard, C. Advanced persistent threats and how to monitor and deter them. Netw. Secur. 2011, 2011, 16–19.
24. Patil, M.R.; Mulimani, C. Pegasus: Transforming Phone Into A Spy. Think India J. 2019, 22, 7883–7890.
25. 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 February 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 176–182.
26. 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.
27. Thomas, T.; Surendran, R.; John, T.S.; Alazab, M. Intelligent Mobile Malware Detection; CRC Press Routledge Publisher: Boca Raton, FL, USA, 2022.
28. Ansar, S.A.; Yadav, J.; Dwivedi, S.K.; Pandey, A.; Srivastava, S.P.; Ishrat, M.; Khan, M.W.; Pandey, D.; Khan, R.A. A Critical Analysis of Fraud Cases on the Internet. Turk. J. Comput. Math. Educ. 2021, 12, 2164–2186.

29. 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.
30. Dhalaria, M.; Gandotra, E. Android malware detection techniques: A literature review. Recent Patents Eng. 2021, 15, 225–245.
31. Stevanoski, G.; Kacurova, M.; Bogatinov, D. Rootkits-cyber security challenges and mechanisms for protection. ETIMA 2021, 1, 174–181.
32. Ramamurthy, M. Fraudster Mobile Apps Detector in Google Playstore. J. Comput. Theor. Nanosci. 2020, 17, 1752–1757.
33. 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.
34. 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.
35. 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.
36. 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.
37. 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.
38. 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.
39. 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.

40. 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.
41. 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.
42. 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.
43. 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.
44. 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.
45. Alrawais, A. Security Issues in Near Field Communications (NFC). Int. J. Adv. Comput. Sci. Appl. 2020, 11.
46. Tu, Y.J.; Piramuthu, S. On addressing RFID/NFC-based relay attacks: An overview. Decis. Support Syst. 2020, 129, 113194.
47. Singh, M.M.; Adzman, K.; Hassan, R. Near Field Communication (NFC) technology security vulnerabilities and countermeasures. Int. J. Eng. Technol. 2018, 7, 298–305.
48. 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.
49. 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.
50. 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.

51. 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.
52. 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.
53. 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.
54. 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.
55. Giri, A. A Study on Efficient Battery Management System Providing Features to Resolve Damage occurring in Mobile Phones.
56. 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.
57. 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.
58. 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.
59. 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.
60. 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.
61. Lin, J.; Seibel, J. Motion-based side-channel attack on mobile keystrokes, 2019.
62. 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.
63. 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.

64. 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.
65. 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.
66. 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.
67. 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.
68. 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.
69. 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.
70. Bo, L.; Fengjun, L.; Guanghui, W.; Wang, L. 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.
71. 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.
72. 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.
73. 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.
74. 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.

75. 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.
76. Kumar, Y. Juice Jacking-The USB Charger Scam. Available at SSRN 3580209 2020.
77. Goodin, D. Hackers Have Been Exploiting 4 Critical Android Vulnerabilities. Ars Technica 2021.
78. 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.
79. 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.
80. 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.
81. Mayrhofer, R.; Stoep, J.V.; Brubaker, C.; Kralevich, N. The android platform security model. ACM Trans. Priv. Secur. 2021, 24, 1–35.
82. 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.
83. 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.
84. 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.
85. Fahrianto, F.; Nurhayati; Kastari. 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.
86. 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.

87. 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.
88. 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.
89. 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.
90. 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.
91. 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.
92. 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.
93. 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.
94. Escobar, F.S.; da Silva, A.S.; Vergara, L.O.C. Nova Vulnerabilidade DO Android. Semin. Technol. Manag. Educ. 2020, 2.
95. Eliassen, K.O. Strandens topologier. K&K-Kultur Klasse 2020, 48, 177–208.
96. Sun, P.; Chen, S.; Fan, L.; Gao, P.; Song, F.; Yang, M. VenomAttack: Automated and Adaptive Activity Hijacking in Android.
97. Kasagiannis, G. Security Evaluation of Android Keystore. Master’s Thesis, University of Piraeus, Pireas, Greece, 2018.
98. 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.
99. 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.

100. 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.
101. 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.
102. 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.
103. 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.
104. 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.
105. 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.
106. 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.
107. 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.
108. 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.
109. 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.
110. 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.
111. 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.

112. 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.
113. 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.
114. Qasmi, W.N.A. Cellular Networks under Signalling Attacks. Ph.D. Thesis, Lahore University of Management Sciences, Punjab, Pakistan, 2019.
115. 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.
116. 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.
117. Mone, G. The quantum threat. Commun. ACM 2020, 63, 12–14.
118. 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.
119. 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.
120. 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.
121. 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.
122. Chess, B.; McGraw, G. Static analysis for security. IEEE Secur. Priv. 2004, 2, 76–79.
123. Landi, W. Undecidability of static analysis. ACM Lett. Program. Lang. Syst. 1992, 1, 323–337.
124. 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.

125. 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.
126. Vamvatsikos, D.; Cornell, C.A. Incremental dynamic analysis. Earthq. Eng. Struct. Dyn. 2002, 31, 491–514.
127. Wong, M.Y.; Lie, D. Intellidroid: A targeted input generator for the dynamic analysis of android malware. Proc. NDSS 2016, 16, 21–24.
128. 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.
129. 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.
130. 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.
131. 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.
132. 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.
133. 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.
134. Bruno, G.; Batina, L.; Bosma, W. Crypto Security Optimizations; Radboud University Nijmegen: Nijmegen, The Netherlands, 2021.
135. 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.

136. 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.
137. 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.
138. 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.
139. 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.
140. 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.
141. 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.
142. 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.
143. Bauer, S.; Rass, S.; Schartner, P. Generic parity-based concurrent error detection for lightweight ARX ciphers. IEEE Access 2020, 8, 142016–142025.
144. 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.
145. 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.

146. 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.

147. 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.

148. 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.

149. 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.

150. 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.

151. Nguyen-Vu, L.; Ahn, J.; Jung, S. Android fragmentation in malware detection. Comput. Secur. 2019, 87, 101573.

152. 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.

153. 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.

154. 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.

155. 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.

156. 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.

157. 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.

158. 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.

159. 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.

160. 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.

161. 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.

162. 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.