electronics

MDPI

Review

A Review of Mobile Surveillanceware: Capabilities, Countermeasures, and Research Challenges

Cosimo Anglano

Dipartimento di Scienze e Innovazione Tecnologica, Università del Piemonte Orientale, 13100 Vercelli, Italy; cosimo.anglano@uniupo.it

Abstract

Mobile smartphones are prime targets for sophisticated surveillanceware, designed to covertly monitor specific individuals. While mobile operating systems implement various protection mechanisms, their defenses are frequently bypassed due to risky user behaviors or underlying software flaws, leading to persistent successful attacks. This paper addresses the critical research problem of how targeted mobile spyware can be effectively counteracted, particularly given its pervasive and evolving threat amplified by sophisticated evasion techniques. To contribute to this understanding, we comprehensively review mobile surveillanceware variants, namely stalkerware and mercenary spyware. We also critically review mobile OS protection mechanisms, and we detail how surveillanceware bypasses or exploits them. Our analysis reveals that, despite continuous efforts by mobile operating system and device manufacturers, both Android and iOS platforms struggle to protect devices and users, particularly against sophisticated mercenary spyware attacks, remaining vulnerable to these threats. Finally, we systematically review state-of-the-art countermeasures, identify their shortcomings, and highlight unresolved research challenges and concrete directions for future investigation for enhanced prevention and detection. Crucially, this future research must increasingly leverage artificial intelligence, including deep learning and large language models, to effectively keep pace with and overcome the sophisticated tactics employed by modern spyware.

Keywords: mobile spyware; surveillanceware; stalkerware; mercenary spyware; malware; spyware prevention; spyware detection; Android; iOS

Academic Editor: Cheonshik Kim

Received: 15 June 2025
Revised: 4 July 2025
Accepted: 7 July 2025
Published: 9 July 2025

Citation: Anglano, C. A Review of Mobile Surveillanceware: Capabilities, Countermeasures, and Research Challenges. *Electronics* **2025**, *14*, 2763. https://doi.org/10.3390/electronics14142763

1. Introduction

Mobile smartphones have become indispensable tools in modern life, serving as central hubs for personal and professional communication, data storage, and access to a vast array of services. This pervasive integration, however, makes them prime targets for *mobile surveillanceware* [1] (referred to simply as *surveillanceware* hereafter), which is a sophisticated form of spyware explicitly designed to covertly monitor and gather sensitive information about users' activities on their smartphones without their knowledge or consent [2].

Surveillanceware [3,4] consists of highly sophisticated tools used to monitor specific individuals or groups, often high-profile targets. These tools employ advanced techniques to evade detection and gather highly sensitive information, with their deployment frequently driven by political motives or the pursuit of high-value data, often involving state-sponsored actors or skilled adversaries. Surveillanceware possesses extensive data collection capabilities, allowing adversaries to gather a wealth of highly sensitive data. This includes, but is not limited to, real-time audio/video from phone calls, microphones,

and cameras, SMS, and instant messaging messages (e.g., WhatsApp, Telegram, Signal), data from various applications (e.g., browsing history, email messages, photos, documents), real-time GPS locations, screenshots, and keystrokes.

Surveillanceware can be further classified into two distinct families, that is *stalkerware* and *mercenary spyware*. Stalkerware is typically employed by individuals for the abusive monitoring of intimate partners [5–9], and relies on relatively simple techniques to keep costs low. Mercenary spyware is instead explicitly designed for espionage, intelligence gathering, or national security objectives by substantially resourced adversaries, such as governmental bodies [3,10–12], and leverages highly sophisticated techniques that necessitate significant financial and technical investments.

Surveillanceware poses an insidious threat to the privacy, and sometimes even the physical safety, of smartphone users. To counter this, mobile operating systems implement various protection mechanisms, such as app isolation, access control, and restricting execution to trusted code. However, despite continuous efforts by OS and device manufacturers to enhance security, successful surveillanceware attacks persist. This ongoing challenge underscores the necessity for continuous research to strengthen defenses.

The pervasive and evolving threat posed by surveillanceware necessitates a comprehensive understanding of its operation and the efficacy of current defenses. This paper aims to address the critical research problem of how surveillanceware can be effectively counteracted, a challenge amplified by its sophisticated evasion techniques.

To lay the groundwork for effective countermeasures, we first characterize the primary capabilities and distinguishing features of surveillanceware, including both stalkerware and mercenary spyware. Following this foundational analysis, the paper critically reviews the protection mechanisms inherent in mobile operating systems designed to safeguard devices and users. A crucial component of this initial assessment involves detailing how these surveillanceware variants are able to bypass or exploit these existing security measures.

This detailed understanding of surveillanceware operational modalities and the successful circumvention of current defensive layers is indispensable for devising robust and truly effective protection mechanisms. Building on this crucial insight into existing gaps, the paper then systematically reviews the state-of-the-art in mobile spyware countermeasures, pointing out their current shortcomings. To conclude, we identify the most pressing unresolved research challenges and propose concrete directions for future investigation, essential for enhancing both prevention and detection capabilities against this insidious threat.

It is important to clarify that this paper solely focuses on surveillanceware. While related, non-discriminatory spyware [13] falls outside of our scope. This latter category—like adware or user-installed apps that gather private info—aims to deploy on a vast number of devices indiscriminately. Its primary goal is broad data collection for advertising or financial gain, prioritizing volume over individual victim identity. For a discussion of the challenges posed by non-discriminatory spyware, please refer to [14–17].

The remainder of this paper is structured as follows. Section 2 describes the capabilities and characteristics of the different forms of surveillanceware. Section 3 describes the protection mechanisms implemented by mobile operating systems. Section 4 details the techniques used by stalkerware, followed by Section 5 which provides an in-depth analysis of mercenary spyware. Section 6 discusses current countermeasures, highlighting research challenges and future research directions in preventing and detecting mobile spyware. Section 7 discusses our findings and their broadest implications and suggests future research directions. Finally, Section 8 concludes the paper.

2. Mobile Surveillanceware: Definition, Capabilities, and Families

Mobile surveillanceware is able to operate covertly in the background, without user knowledge or consent, to stealthily access and transmit to an adversary third-party (the *adversary* in the following, for brevity) potentially all the data generated, stored, and transmitted by the smartphone, including those temporarily stored in its volatile memory.

As anticipated in Section 1, it can be classified into two distinct families, that is, stalkerware and mercenary spyware, which are characterized by different specific intents, goals, and the financial resources of its customer base. These differences translate into varying levels of sophistication in the methods, techniques, and tactics they employ, as discussed below.

2.1. Stalkerware

In recent years, the use of stalkerware apps has grown consistently, paralleled by an expanding commercial market for these tools sold to a general audience [4,18]. While often deceptively marketed for legitimate purposes like parental control or theft recovery, these apps actually enable the hidden, non-consensual surveillance of a phone's activities, location, and communications [5,8].

To broaden their market reach, stalkerware apps prioritize cost containment over sophistication. They are typically sold as moderate-cost subscriptions, usually ranging from USD 50 to USD 150 per month. For this reason, they typically rely on simple and unsophisticated techniques to achieve installation on and monitoring of the target device. In particular, their installation requires physical access to the smartphone, and monitoring is based on the abuse of legitimate access control mechanisms implemented by the operating system.

2.2. Mercenary Spyware

In recent years, the use of mercenary spyware has also significantly increased, accompanied by a fast-growing market and numerous high-profile scandals that have garnered widespread media attention. The mercenary spyware market was estimated to be worth over USD 12 billion in 2023, indicating a lucrative and expanding industry driven by global demand for its sophisticated tools [19,20].

Mercenary spyware is explicitly designed for espionage, intelligence gathering, covert surveillance, or national security objectives by substantially resourced adversaries, such as governmental bodies, law enforcement agencies, or other similar entities [3,21].

The main goal of mercenary spyware is to achieve installation and monitoring without needing physical access to the target device. For this reason, they use very sophisticated techniques that enable highly customized attacks, tailored to the specific target. The design and implementation of these techniques necessitate the substantial investment of both financial and technical resources to keep up with the evolving protection mechanisms implemented by smartphone manufacturers. The resulting cost for their end users is very high. For instance, leaked commercial proposals for a leading mercenary spyware platform priced the spyware management platform and 100 smartphone infections at EUR 8,000,000 [3], and EUR 900,000 for 100 additional smartphone infections (equating to EUR 9000 per infected device) [22].

2.3. The Spyware Kill Chain

Mobile surveillanceware achieves its objectives by executing a precise *kill chain*, a sequence of activities designed for surreptitious installation and continuous user monitoring. This process occurs without the target's knowledge or consent, and despite the mobile operating system's inherent protection mechanisms. In particular, the kill chain, which is graphically depicted in Figure 1, consists of the following steps:

1. *Delivery and exploitation*: Surveillanceware needs first to get on the device and to be installed (*Delivery*). Then, it needs to obtain the required permissions on the device in order to access the data of interest (*Exploitation*). A variety of methods are typically used, which depend on its sophistication. In particular, stalkerware relies on manual installation and the granting of the necessary permissions, so the adversary needs physical access to the device to carry out this stage of the kill chain. In contrast, mercenary spyware typically provides remote delivery and installation through the sending of suitably crafted data to some app or service running on the smartphone, and achieves exploitation by abusing the software vulnerabilities of the operating system and/or of some apps of the device;
2. *Data gathering*: After installation, surveillanceware runs in the background, collecting information and data from the device by leveraging various methods, ranging from merely reading the files storing data of interest to more sophisticated ones like *function hooking* (see Section 5.2 for more details);
3. *Hiding*: To avoid being detected during its operation, surveillanceware hides its presence by exploiting a combination of different methods, ranging from hiding app icons to code obfuscation/encryption, in-memory-only execution, and self-deletion;
4. *Persistence*: After initial installation, after installation, surveillanceware runs in the background, collecting information and data from the device by leveraging various methods, ranging from merely reading the files storing data of interest to more sophisticated ones like *function hooking* needs to find a way to continue its operations across reboots and other interruptions. Persistency techniques vary from auto-restart after a reboot to re-exploitation and re-installation;
5. *Data exfiltration*: Surveillanceware sends collected data to a remote system under the control of the adversary. Also in this case, used methods span from the direct and immediate sending to a publicly visible server to condition-based delivery to anonymized servers.

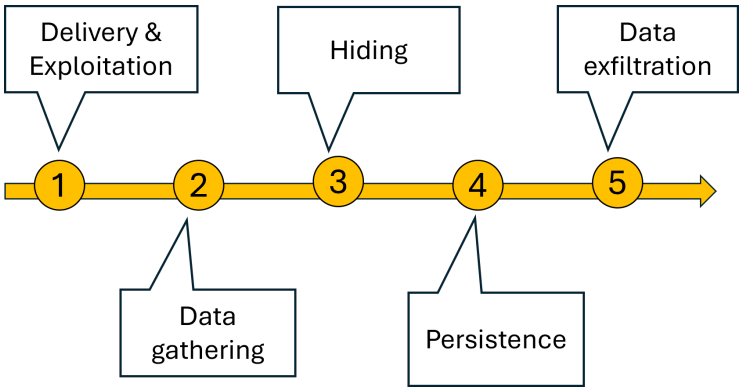

Figure 1. The mobile surveillanceware kill chain.

2.4. Surveillanceware Feature Summary

Table 1 summarizes the characteristics of contemporary stalkerware and mercenary spyware, including notable examples of these apps and platforms. For each category, the table details the following:

- Installation and exploitation method: characterized as manual (Android stalkerware), unnecessary (iOS stalkerware), or remote (mercenary spyware);
- Data collection method: involving either the exploitation of legitimate operating system mechanisms (stalkerware) or software vulnerabilities for privilege escalation (mercenary spyware);

- Scope of data collection: either partial (stalkerware), where specific information types are not gathered, or complete (mercenary spyware), enabling the acquisition of all data stored or real-time generated by the smartphone;
- Typical cost: moderate (in the range of 500–1000 USD/year) for stalkerware, or very high (several millions of US dollars) for mercenary spyware.

Table 1. Features of surveillanceware apps and platforms.

Type	Examples	Installation and Exploitation	Data Collection	Scope of Data	Cost
Stalkwerware	Spyzie, Cocospy, TheTruthSpy, mSpy, FlexiSPY, iSpyoo.	Manual, Unnecessary	Exploitation of legitimate OS mechanisms	Partial	Moderate
Mercenary spyware	Epeius, Pegasus, Predator, Heliconia, Graphite.	Remote	Privilege escalation through software vulnerability exploitation	Complete	Very high

3. Mobile OS Protection Mechanisms Analysis

Mobile operating systems are the foundational software that manage smartphone hardware and software resources, and currently the global market is dominated by just two players: *Android* and *iOS*. As of April 2024, Android holds the vast majority with about 74% market share, while iOS accounts for about 26% [23].

Android is an open source mobile operating system developed by Google, primarily based on a modified version of the Linux kernel. It is known for its extensive customization options, allowing manufacturers and users significant flexibility in interface and functionality. Its app ecosystem, centered around the *Google Play Store*, offers a vast array of applications, making it highly versatile for various tasks and user preferences. Android security is based on a multi-layered architecture built upon the Linux kernels [24], primarily relying on application sandboxing, a robust permissions model, and hardware-backed security features to protect user data and device integrity, combined with continuous software updates and its real-time scanning.

iOS is the Apple's proprietary mobile operating system. It is characterized by a tightly controlled ecosystem, featuring a robust security and privacy model, with stringent App Store review processes, hardware-backed encryption, and advanced biometric authentication. iOS security [25] is based on a tight hardware–software integration, and on the use of mechanisms like secure boot, code signing, strong encryption tied to hardware and user credentials, application sandboxing, and user-centric privacy control, all enforced by the operating system and backed by dedicated hardware. Regular, unified software updates are pushed directly by Apple, providing consistent new features and critical security patches to all supported devices promptly.

To protect users from malware in general, and from surveillanceware in particular, these operating systems implement various mechanisms that work in synergy among them to prevent the successful completion of one or more steps of the kill chain. These mechanisms operate on the principle of assigning limited privileges to smartphone users and preventing privilege escalation. This ensures that applications also run with restricted permissions, preventing them from accessing sensitive hardware resources like the microphone or camera, or confidential data stored by other applications. In particular, the following mechanisms are used:

- *Application isolation*: To prevent applications from accessing each other's data, each app is confined within its own *sandbox*. This isolated, secure environment limits the access of the app to system resources and other application data, effectively preventing malicious or misbehaving software from compromising the entire device or user privacy;
- *Application access control*: To prevent unrestricted and uncontrolled access to system-wide hardware resources (e.g., the microphone and the camera), as well as to sensitive system data (e.g., the locations stored by the GPS sensor), mobile operating systems employ robust access control mechanisms which govern which applications or system processes can access specific resources or perform particular actions;
- *Execution restricted to trusted code*: To avoid the execution of malicious code, mobile operating systems implement mechanisms that allow only *trusted code* to run, i.e., code that is assumed to be reliable, secure, and to operate as intended without causing harm or unintended consequences. Trusted code execution is enforced by the following mechanisms:
  - *App store protection*: The primary source for the distribution of applications to devices is restricted to curated app stores as (i.e., *Google Play* for Android and *Apple Store* for iOS), whereby developers publish their apps on the store, and users can download and install them from there. This enables the manufacturer of the operating system to extensively check the security of any app before it is made available to devices;
  - *Code signing*: Trusted apps are digitally signed by the corresponding developers with their private keys. The signature of a given app is then verified when that app is installed, updated, or even executed. In this way, the operating system can check that the app originates from the stated developer and has not been tampered with after installation;
  - *Secure boot*: To ensure that only the trusted operating system code is run on the smartphone, the boot process is secured by using code signing in such a way to establish a hardware-based chain of trust. In particular, the boot process is divided into a sequence of stages, with each stage cryptographically verifying the integrity and the authenticity of the following one before executing it. The first stage (the *boot ROM*) is directly embedded into the hardware of the device, and is therefore immutable. This prevents malicious software from running during startup and compromising the device's security.

Despite significant efforts by mobile operating system manufacturers to enhance security mechanisms and keep smartphones safe, their effectiveness in achieving expected levels of protection can be compromised by the two following problems.

The first problem consists of risky behaviors carried out by users, which may undermine the code security and application isolation mechanisms:

- *Device privilege elevation*: Smartphone users may elevate their privileges to the highest level by exploiting technical procedures (named *rooting* and *jailbreaking* for Android and iOS, respectively) which leverage software vulnerabilities. As a consequence, apps are executed with the highest privilege level too, and may therefore evade sandboxing;
- *App over-permissioning*: An app may obtain access to other apps or system data if the user grants it excessive permissions to that app;
- *App side-loading*: It consists of the installation of apps from unofficial sources rather than from official app stores. Side-loading bypasses checks for malicious behavior, code integrity, and trusted origins, allowing surveillanceware to masquerade as safe applications. Side-loading is allowed both by Android and iOS, but the former provides a less restrictive and simpler method than the latter.

The second problem, conversely, stems from vulnerabilities within the operating system or application code. If successfully exploited, these vulnerabilities can be leveraged to bypass all security measures, enabling the remote installation and execution of surveillanceware, evasion from the sandbox, and data exfiltration. For this reason, great efforts are being made by mobile operating systems, device manufacturers, and the cybersecurity research community, to detect and patch software vulnerabilities as quickly as possible. Vulnerabilities are said to be *one-day*, meaning that they have been disclosed (disclosed vulnerabilities are stored into the *CVE database* [26], and each one of them is identified by its unique *CVE ID* (e.g., *CVE-2025-1234*, which identifies the 1234th vulnerability that has been added in the year 2025).) and there has been "one day" (or more) of awareness and potential mitigation, while those that are unknown/unpublished yet are said to be *zero-day*, meaning that they are unknown, therefore leaving zero days to patch them before exploitation).

4. Stalkerware Analysis

Stalkerware apps implement their kill chain using unsophisticated techniques, a direct result of their marketing strategy to offer relatively low-cost products.

In the past, the simplest way for stalkerware to bypass operating system protections and streamline the entire surveillanceware kill chain was through rooting or jailbreaking the victim's smartphone. Consequently, early stalkerware installation procedures often required users to root/jailbreak the target device.

However, rooting and jailbreaking are technically complex and typically beyond the average individual's technical capabilities. Moreover, these procedures are not available for all devices or operating system versions, especially within the iOS ecosystem.

For these reasons, contemporary stalkerware no longer relies on rooting or jailbreaking. Instead, it now abuses regular mobile operating system mechanisms. While these techniques offer limitations in the amount of data gatherable compared to a rooted or jailbroken device (as they do not allow sandbox evasion), they still enable sufficient data collection for surveillance purposes.

Given the distinct architectures of Android and iOS, stalkerware apps utilize different implementation techniques on each platform. Table 2 overviews these methods, presenting a high-level description of the techniques typically used for each kill chain stage on Android and iOS. A detailed review of these specific methods, based on extensive technical analysis literature [4,5,8,9,27–31], follows in the remainder of this section.

Table 2. Stalkerware kill chain implementation methods.

OS	Delivery and Exploitation	Data Gathering	Hiding	Persistence	Data Exfiltration
Android	Installation via side-loading, exploitation via over-privileging the app (Section 4.1.1)	Leverage of legitimate OS mechanisms to access data of other apps and system hw resources for real-time monitoring (Section 4.1.2)	Concealment of app icon (Section 4.1.3)	Obscuration of uninstallation procedure and/or implementation as "diehard" service (Section 4.1.3)	Transmission via email or upload to remote server (Section 4.1.3)
iOS	Installation not needed	Exploitation of *iCloud backups* (Section 4.2.1) or of *Wi-Fi backups* (Section 4.2.2)	Unnecessary (no installation)	Unnecessary (no installation)	Access to remote backups

4.1. Android Stalkerware

Android's open and flexible nature, which allows app sideloading and granular permission control, creates a relatively large attack surface for stalkerware. Installing such software typically requires brief physical access to the target device to enable unknown sources or grant dangerous permissions for monitoring activities, location, and communications. This facilitates various methods for stalkerware to implement its kill chain [5,27,28,32], which are discussed in detail below.

4.1.1. Delivery and Exploitation

Stalkerware is typically delivered to a target Android device by exploiting the operating system's native side-loading mechanism, usually by downloading the app via a web browser on it. This process requires physical access to the device and its unlocking, since installation needs to be performed manually after some system settings have been modified (e.g., enabling the *Install Unknown Apps* setting and disabling the *Google Play Protect* functionality).

Instead, exploitation occurs through the manual granting of specific permissions to the stalkerware app [28]. These include

- *Dangerous permissions*, which enable control over core device functions (like the microphone, camera, keyboard, and screen) and access to private user data (such as geographical position, call history, contacts, SMS messages, and potentially files on shared storage);
- *Accessibility services*, which allow the stalkerware app to interact with the user interface, read screen content, and capture user inputs;
- *Notification access*, which provides the stalkerware app with the ability to view the text of notifications generated by other applications.

4.1.2. Data Gathering

Stalkerware apps are able to achieve extensive data-gathering capabilities by abusing operating system mechanisms, which are used for purposes other than their original design, after they have been over-permissioned during their installation. In particular, data gathering capabilities are implemented as follows:

- *Accessing data of other applications*: Data belonging to other applications is gathered indirectly through several techniques. Accessibility services are leveraged to gather data rendered on the screen by any app the user is running [33] or to take screenshots [28]. Data are also gathered by reading the notifications that apps send to users, such as those notifying incoming messages [28]. Keystrokes are gathered by registering the stalkerware app as a listener for keyboard events or changes in UI elements like text fields;
- *Camera surveillance*: To covertly take pictures, record videos, or stream live videos, stalkerware commonly employs a few techniques [28]. These include
  - Rendering the camera preview window imperceptible by setting its size to a minimal (e.g., 1×1 pixel) or transparent state;
  - Intercepting raw camera frames directly using specific Android API functionalities, bypassing any preview display;
  - Using an invisible 1×1 pixel in-app browser to stream live videos at full resolution via a dynamically loaded JavaScript version of the *WebRTC* framework (the *WebRTC* (web real-time communication) is an open source framework that enables web browsers and mobile applications to perform real-time, peer-to-peer audio, video, and data communication directly, without requiring plugins or intermediary servers.);

- *Phone call recording*: To record a phone call, both uplink and downlink audio must be captured. The uplink audio is directly recorded via the microphone. Downlink audio is instead gathered by activating the speaker and disabling its noise-canceling feature, which permits the microphone to pick up the audio emanating from the device's speaker;
- *Voice call recording*: To record calls made with third-party applications (e.g., WhatsApp or Viber), downlink audio is captured using the same technique as for phone calls. Uplink audio is instead acquired by concurrently accessing it alongside the third-party app (which is a legitimate Android functionality). Additionally, side-channels, such as standard notifications or accessibility actions, are exploited to detect when calls are active.

4.1.3. Hiding, Persistence, and Data Exfiltration

Stalkerware primarily achieves hiding by concealing its app icon. This is programmatically achieved by setting a specific app attribute and by exploiting the mechanisms designed to launch the app even when its icon is hidden.

Persistence is instead achieved by leveraging two distinct mechanisms, which are often used in combination. First, the uninstallation process is obscured to prevent easy removal by locking the device during an uninstallation attempt, and abusing accessibility services to block confirmation clicks or dismiss prompts. Second, the stalkerware app often installs itself as a "diehard" service which automatically restarts after being stopped or terminated by the system, or after device reboots.

Finally, collected data is exfiltrated either by direct transmission to an adversary-controlled email address or by uploading it to a remote server maintained by the stalkerware vendor, which provides a dashboard for the adversary to peruse the data.

4.2. iOS Stalkerware

On non-jailbroken iOS devices, stalkerware face greater challenges than on Android in abusing legitimate operating system mechanisms to implement its kill chain. Without a jailbreak, bypassing sideloading, code-signing, and achieving sandbox evasion is indeed impossible. Apple's "walled garden" approach, characterized by strict App Store controls and rigorous review, significantly hinders unauthorized app distribution. Furthermore, robust iOS sandboxing tightly limits app access and background operations. This combination makes it very difficult for typical stalkerware to be installed or function, unlike on Android [34,35].

For these reasons, current iOS stalkerware does not rely on installing a malicious app directly on the smartphone. Instead, it exploits access to data that has already legitimately left the device to be stored elsewhere. Specifically, currently used techniques abuse iOS's backup functionality, either by accessing data stored in iCloud backups or by creating a backup on a computer connected to the same Wi-Fi network. In essence, iOS stalkerware implements only the *data gathering* and *data exfiltration* steps of the kill chain, as on-device installation is bypassed.

4.2.1. Data Gathering via iCloud Backups

iCloud Backup is an iOS feature that automatically creates a secure copy of a device's data and settings, storing it on the *iCloud* service associated with the *Apple ID* used on that device. If an adversary gains knowledge of the *Apple ID* and its corresponding password, they can access the *iCloud* backups and all the data within them.

This feature is exploited by stalkerware as follows. The stalkerware vendor runs a service, on a remote computer it controls, which authenticates with the victim's credential on the *iCloud* service, and downloads the backup files stored there. Essentially, the stalkerware

acts as an intermediary, pulling data from Apple's servers using the victim's credentials and making it accessible to the adversary.

In principle, this method could enable extensive data gathering. However, it faces several significant challenges posed by iOS's security policies and mechanisms. Specifically, *iCloud* now mandates two-factor authentication (2FA), which stalkerware services cannot technically disable or bypass. While theoretically possible to satisfy the 2FA prompt (e.g., via physical access to a trusted device or social engineering), this remains practically difficult. Furthermore, *iCloud* backups may be incomplete, as users can disable the inclusion of certain data categories and third-party app developers can opt out of backup inclusion. Moreover, since January 2023, end-to-end encryption can be enabled for *iCloud* backups, storing the decryption key solely on the user's smartphone, making thus these data inaccessible to stalkerware. Lastly, data within *iCloud* backups is inherently latent and infrequent, reflecting past device states and updating approximately only once every 24 h under specific conditions (Wi-Fi and power source).

4.2.2. Data Gathering via Local Wi-Fi Synchronization

Another method for indirectly gathering data from an iOS device involves exploiting its local Wi-Fi synchronization feature, which allows a device to sync data and create backups with a designated computer over a shared Wi-Fi network. Stalkerware leverages this by using a controlled computer running specialized software that either mimics the legitimate sync process or intercepts backup files created during a Wi-Fi sync session.

This mechanism requires an initial setup phase, in which the target device must be connected to the adversary's computer via USB, the computer pairing authorized, and the Wi-Fi synchronization option enabled. This means that the adversary needs temporary physical access to the target device. Once this initialization is successfully completed, whenever the target iOS device connects to the same Wi-Fi network as the adversary's pre-configured computer (and potentially meets other conditions like being plugged into power), the stalkerware desktop component can initiate or receive a wireless backup of the device data. These data are then uploaded to a remote server controlled by the stalkerware vendor, making it accessible to the adversary.

Similarly to the *iCloud* backup method, Wi-Fi synchronization presents several challenges for stalkerware. In particular, initial physical access is strictly mandatory to authorize device pairing with a computer and to enable Wi-Fi synchronization. Furthermore, Wi-Fi synchronization suffers from the same data latency and infrequency issues that characterize the *iCloud* backup method, meaning data reflects past device states and updates are not real-time or continuous.

5. Mercenary Spyware Analysis

As mentioned in Section 2.2, mercenary spyware is explicitly designed for the covert surveillance of high-profile targets, prioritizing maximum stealth and data-gathering capabilities. To achieve this, it employs sophisticated technical solutions enabling remote delivery and installation (eliminating physical access), unrestricted access via OS/app vulnerabilities, covert hiding and persistence through advanced, hard-to-detect techniques, and data exfiltration via complex anonymization infrastructures to evade detection and attribution.

Mercenary spyware is typically deployed as a sophisticated, multi-component system, which includes the following components:

- The *Infection vector/Deployment mechanism*, whose function is to initially gain access to the target device;
- The *Agent*, which is the core malicious software residing on the target device with the purpose of gathering data;

- The *command-and-control (C2) infrastructure*, which is the nerve center that allows the attacker to communicate with and control the agents on infected devices;
- The *anonymization/obfuscation infrastructure*, which leverages various techniques to make the adversary undetected and untraceable.

In this section, we review the technical methods and techniques used by mercenary spyware systems to implement their kill chain. Our review is based on the technical analysis of several widely known mercenary spyware platforms currently in use, namely *Epeius*, *Pegasus*, *Predator*, *Graphite*, and *Graphite* (see Table 1 in Section 2.4).

Table 3 outlines the common implementation techniques employed across the kill chain stages. These methods are platform-agnostic, functioning identically on Android and iOS due to their shared underlying mechanics. Furthermore, the table makes no distinction between different mercenary spyware platforms, as they all utilize specific instantiations of the general techniques presented.

Table 3. Mercenary spyware kill chain implementation methods.

Delivery and Exploitation	Data Gathering	Hiding	Persistence	Data Exfiltration
Complex chains of software exploits featuring combinations of *zero-click*, *one-click* and *network injection* exploits (Section 5.1)	*Direct file access* and/or *function hooking* (Section 5.2)	Injection into legitimate processes, in-memory-only execution, encrypted payload, erasure of activity traces, self-destruction (Section 5.3)	Re-execution of delivery and exploitation, installation on system restricted file systems (Section 5.4)	Data encrypted and routed through an anonymization infrastructure (Section 5.5)

Due to the significant barriers to academic research on mercenary spyware, including the lack of public access to samples, our analysis is largely informed by reports from international organizations and companies [22,36–41] and by reportedly leaked documentation [42–45], filling a gap left by limited academic publications [10–12].

5.1. Delivery and Exploitation

To achieve delivery and exploitation, mercenary spyware typically employs complex exploit chains that link multiple vulnerabilities for full device compromise. These chains often begin by exploiting an initial flaw to gain code execution within a sandboxed process, commonly targeting applications or system services that receive network data (e.g., messaging apps like WhatsApp and iMessage, or Internet browsers). This initial exploit is then followed by subsequent exploits aimed at sandbox evasion and privilege escalation, targeting the operating system kernel either directly or through system services or libraries.

Typically, the initial exploit is delivered through one of the following methods:

- *Zero-click* exploits: The exploit code is sent to a vulnerable app or service running on the device, where it autonomously starts its execution, without any explicit action or interaction carried out by the user. Examples are provided by *Pegasus* and *Graphite*, which leverage zero-click exploits against WhatsApp and iMessage triggered by specially crafted messages [3,38,46]);
- *One-click* exploits: the initial exploit is downloaded onto the target device when the victim is tricked into clicking a malicious link, typically delivered via message (SMS, WhatsApp, etc.). This method is less effective than zero-click exploits, as vigilant users can simply avoid clicking the malicious link. Despite this limitation, one-click attacks are utilized by most mercenary spyware [3,10,36,46];

- *Network injection* exploitation: It involves intercepting and modifying network traffic directed at the target device to inject the initial exploit. This method allows the exploit to be downloaded without any user action, effectively transforming a one-click exploit into a zero-click one. While network injection typically requires the cooperation of the victim's ISP or mobile carrier [22,42–44], knowing the target phone number can enable the use of tactical network elements—devices mimicking legitimate base transceiver stations—to remotely install the initial exploit of the chain [47].

To better illustrate how the exploit chains used by mercenary spyware function in practice, we describe two recent notable examples: the first targeting Android, and the second targeting iOS.

5.1.1. Example of an Android Exploit Chain

The first example we consider is the exploit chain used by the *Heliconia* spyware on Samsung Android devices, which has been discovered in December 2022 [3,48].

The concatenation of the exploits used by this chain is graphically shown in Figure 2, and consists of the five following stages:

- *Stage 1—initial access*: the first stage of the chain is delivered through a one-click exploit of a zero-day vulnerability (*CVE-2022-4262*) in the Samsung Browser, which allows the spyware to run inside the browser process. The exploit is triggered by the click on a malicious link sent to the victim;
- *Stage 2—sandbox escape*: An unpatched one-day vulnerability of the Samsung Browser (*CVE-2022-3038*) is exploited to escape its sandbox;
- *Stage 3—privilege escalation*: Another unpatched one-day vulnerability in the kernel driver of the *Mali* GPU (*CVE-2022-22706*) (which is used by Samsung smartphones) is exploited to escalate privileges;
- *Stage 4—kernel read and write access*: a second zero-day vulnerability in the Linux kernel sound subsystem (*CVE-2023-0266*) is exploited to gain kernel read and write access;
- *Stage 5—installation*: The vulnerabilities exploited in stages 3 (*CVE-2023-0266*) and 4 (*CVE-2022-22706*) are exploited again to obtain the final installation of the *Heliconia* spyware.

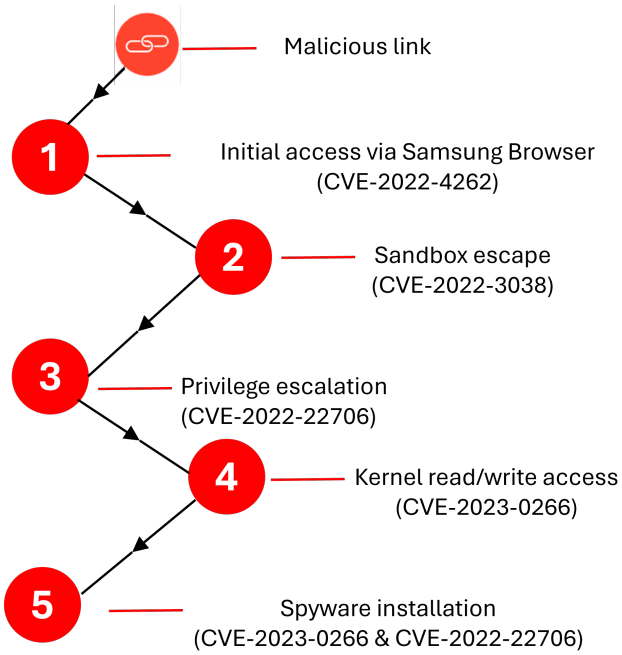

Figure 2. The Heliconia Android exploit chain.

5.1.2. Example of an iOS Exploit Chain

The second example we consider is the exploit chain used by the *TriangleDB* spyware on smartphones running iOS up to version 16.2 which was discovered at the end of 2023 [49], and is arguably the most complex exploit chain discovered so far, featuring four zero-days vulnerabilities [50].

The exploit chain concatenation of exploits, graphically depicted in Figure 3, consists of the seven following stages [51]:

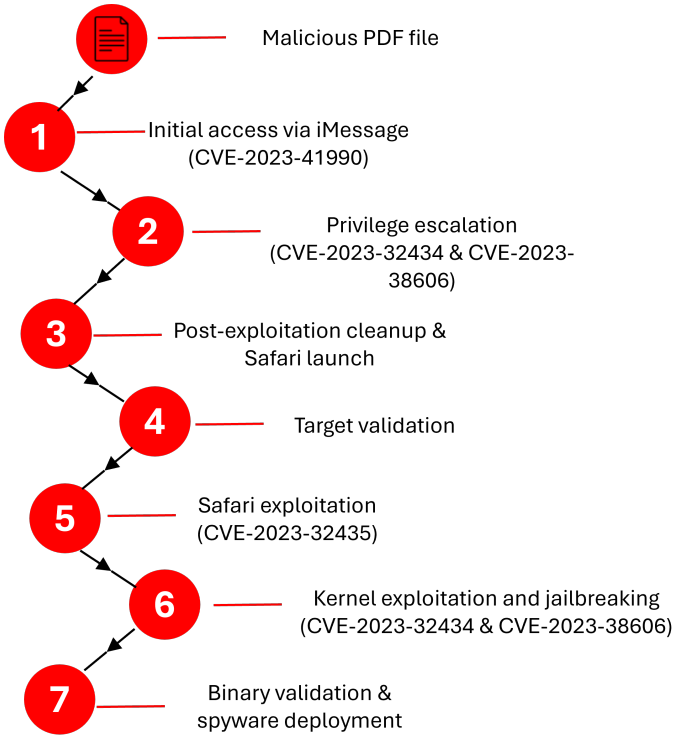

Figure 3. The operation triangulation exploit chain (adapted from [51]).

- *Stage 1—Initial access*: The first stage of the chain is delivered through a zero-click zero-day vulnerability of the *iMessage* system app (*CVE-2023-41990*), and is triggered by an invisible message containing a malicious PDF file as attachment, which allows remote code execution of the first exploit of the chain;
- *Stage 2—Privilege escalation*: A second zero-day vulnerability in the iOS kernel (*CVE-2023-32434*) is exploited to gain read/write access to the entire physical memory of the device, followed by the exploitation of a third zero-day vulnerability also in the iOS kernel (*CVE-2023-38606*) to disable several hardware memory protection mechanisms;
- *Stage 3—Post-exploitation cleanup and Safari launch*: Some cleanup actions are performed to cover the tracks of the exploitation by launching a legitimate OS process and injecting it into code, removing the traces of initial exploit delivery. Then, a Safari process in invisible mode is launched;
- *Stage 4—Target validation*: The invisible Safari browser downloads, from a website controlled by the adversary, a JavaScript program that extensively fingerprints the device to verify that it is the intended target of the spyware. If the checks are passed, the subsequent stage of the chain is executed, otherwise the chain is aborted;
- *Stage 5—Safari exploitation*: The invisible Safari browser loads another web page containing an exploit of a fourth zero-day vulnerability in the *WebKit* engine of the browser (*CVE-2023-32435*), which enables the execution of malicious code within the context of the Safari process;

- *Stage 6—Kernel exploitation and jailbreaking*: The exploited Safari process runs the same exploits used in stage 2 to achieve jailbreaking of the device;
- *Stage 7—Binary validation and spyware deployment*: Finally, a binary validator program runs to
  - Remove exploitation traces from system databases;
  - Detect whether the device is already jailbroken (potentially indicating a research device);
  - Gather extensive device information to ensure that it is the actually targeted smartphone;
  - Load the *TriangleDB* spyware [49].

5.2. Data Gathering

After obtaining kernel read/write permissions, mercenary spyware can gather both data stored in the internal memory of the device and real-time data generated by its hardware resources.

Two methods, often used in combination, are typically employed to gather the data stored on the device, as schematically depicted in Figure 4.

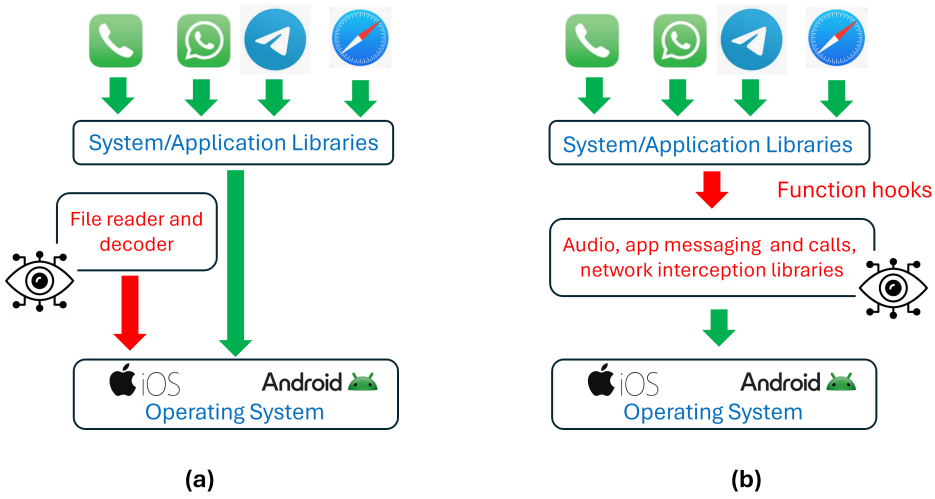

Figure 4. Mercenary spyware data-gathering techniques: (**a**) Direct file reading; (**b**) Function hooking.

The first method (Figure 4a) involves directly reading and parsing files stored on the device to extract meaningful information, such as messages, contacts, browsing history, authentication credentials, etc. A *File reader and decoder* module, running with kernel privileges, handles this data acquisition and decoding.

The second method (Figure 4b) involves *function hooking* [52], a technique where the execution flow of a target function is redirected to a custom function, typically by modifying its entry point in memory. Specifically, the spyware hooks its own custom functions to system or library functions utilized by various applications—such as those handling message transmission, data encryption/decryption, user input, or network requests. This allows the spyware to inspect and decode all data passed by an app to these hooked functions. After gathering the information of interest, the custom spyware function then calls the original function to avoid altering the expected behavior of the running application.

Function hooking offers a stealthier approach for richer, real-time data but is more complex and potentially less stable than direct file access. Conversely, direct file reading is simpler but can be noisier, miss real-time data, and often requires parsing complex formats. For these reasons, mercenary spyware commonly combines these techniques to maximize

data acquisition while minimizing detection. For instance, hooking might capture real-time messages, while direct file access retrieves historical logs.

Finally, the *Graphite* spyware recently demonstrated a third data-gathering technique: it silently loads into existing, legitimate application processes by exploiting application-level vulnerabilities, directly accessing relevant data from their memory. This method does not require root privileges, only exploitable vulnerabilities within the target applications, but it grants full access solely to the data of those exploited applications.

To access the data generated by hardware resources—such as the camera, microphone, display, GPS receiver, and various smartphone sensors—mercenary spyware directly interacts with their corresponding kernel drivers. This grants it the capability to activate/deactivate these resources and gather the data they generate.

5.3. Hiding

To hide its presence on a device, mercenary spyware employs several tactics to elude detection.

The first type of tactic aims to prevent its operations from being noticed by users or detection programs. Employed techniques include injecting itself into legitimate processes (e.g., *Graphite*), running only in memory without creating executable files on the device (e.g., *Predator* and *Pegasus*), delivering encrypted payloads (e.g., *Pegasus*), and erasing traces of its activity (e.g., *Predator* and *Pegasus*).

The second type of tactic aims to prevent or hinder the study of spyware behavior, thus preventing its characterization for developing detection techniques. Mercenary spyware often includes routines to detect whether it is running in an analysis environment (like a device emulator) or if the device shows signs of forensic tool usage. In such cases, it typically terminates execution and triggers a self-destruction mechanism.

5.4. Persistence

As an initial step to ensure persistence, mercenary spyware typically disables firmware updates, preventing the device from receiving security patches that could remove the spyware or its exploited vulnerabilities.

Beyond this, two main persistence tactics are observed. For mercenary spyware designed to run only in memory, delivery and exploitation are re-executed after each reboot. This approach is most effective when the exploit chain begins with a zero-click exploit, as repeatedly tricking a user into clicking a malicious link can be challenging. If, however, the spyware needs to install an executable on the device, it typically modifies the device's configuration to install itself onto file systems restricted to the operating system. From these locations, the spyware is launched with each reboot, often by installing itself as a system application that starts during the boot process, or by replacing legitimate system programs that launch at boot time.

To illustrate, let us review the methods employed by *Predator* to achieve persistence after each reboot, starting from executable code covertly stored on the device:

- *Android devices*: It injects itself into the *Zygote* system process, and in this way it gains a highly privileged position from which to monitor and manipulate other applications [36]. This is a fundamental part of the Android runtime system, and is the first Java Virtual Machine (JVM) process started at boot and serves as the parent for nearly all other application processes. When a new application is launched, the Zygote process forks itself to create a new process for that application, sharing core libraries for efficiency;
- *iOS devices:* It leverages the *iOS shortcuts automation* mechanism to re-run the exploit code whenever one of a predefined list of commonly used applications is launched by the user [53,54].

5.5. Data Exfiltration

Mercenary spyware exfiltrates data by encrypting it and routing communications through multiple layers of anonymizing proxies and servers to avoid attribution. This contrasts sharply with stalkerware, which typically sends collected data directly to remote servers.

To ensure both resilience and operator obfuscation, mercenary spyware infrastructures commonly use multiple layers of domains and servers, often leveraging cloud hosting [22]. They also employ techniques aimed at avoiding the translation of the IP addresses of servers into their fully qualified domain names (e.g., DNS lookups). The anonymization of their *C2* infrastructure is vital for the operational longevity of mercenary spyware campaigns. By utilizing multi-layered proxy networks and dynamically shifting infrastructure (e.g., frequently changing *C2* server IP addresses, domains, and hosting providers), adversaries significantly complicate attribution and efforts by law enforcement or security vendors to disrupt their operations.

6. Proposed Countermeasures: State of the Art and Research Challenges

As noted in Section 3, despite sophisticated protection mechanisms implemented by operating systems, successful surveillanceware attacks are continuously documented for both stalkerware [55–57] and mercenary spyware [58,59]. The reasons for this persistent issue, explored in detail in Sections 4 and 5, stem from a combination of user inattention and the presence of software vulnerabilities, which together enable surveillanceware to successfully implement its kill chain. This challenge is further compounded by an inherent asymmetry between attacker capabilities and the defensive posture of an average user, especially when confronting highly sophisticated mercenary spyware.

This situation calls for a multi-layered defense strategy, combining robust technological safeguards with vigilant user practices and proactive security measures, which encompasses the three following levels:

- *Prevention*: Proactive measures designed to block the successful completion of the kill chain;
- *Detection*: Identifying and alerting about the potential presence of surveillanceware that has bypassed preventive measures as early as possible to minimize its impact.
- *Avoidance*: Proactive risk management strategies aimed at eliminating exposure to surveillanceware compromise by choosing not to engage in risky activities or use threatening technologies, thereby sidestepping potential risks altogether.

Together, these levels work synergistically: avoidance reduces the overall attack surface, prevention identifies and fixes as many vulnerabilities as possible, and detection catches what prevention misses, providing crucial intelligence to refine both protective and avoidance strategies in a continuous feedback loop.

Prevention and detection are grounded in technical solutions and engineering principles, which we discuss throughout the remainder of this section, where we examine the state-of-the-art in these areas, highlight their shortcomings, delve into unresolved research challenges, and propose directions for future investigation.

In contrast, avoidance techniques are behavioral strategies focused on educating users to steer clear of risky actions. These techniques are outside the scope of this paper, but interested readers can find a thorough exploration in [60–62].

Table 4 outlines the frontiers and open challenges for surveillanceware prevention and detection, which we expand upon in the remainder of this section. As the table shows, prevention requires both the ability to fix software vulnerabilities and to strengthen operating system mechanisms. Achieving these goals involves solving various challenges, including developing better identification and patching/rollout solutions, and establishing

novel paradigms that enable operating systems to balance essential user functionality with reduced privacy risks.

The table also highlights that effective detection necessitates analyzing observable data from suspect applications or services. While current methods leverage the analysis of application structure, generated data, or observed behavior, none are effective in isolation. Consequently, the present challenge is to develop machine learning (ML) and deep learning (DL)-based approaches capable of integrating all these data streams into a unified predictive model.

Table 4. Surveillanceware prevention and detection approaches.

Prevention (Section 6.1.1)		**Detection (Section 6.2)**			
Vulnerability fixing (Section 6.1.1)	OS mechanisms strengthening (Section 6.1.2)	Signature-based (Section 6.2.1)	IoC-based (Section 6.2.2)	Behavior-based (Section 6.2.3)	ML/DL-based (Section 6.2.4)

6.1. Proposed Surveillanceware Prevention Methods: Frontier and Open Challenges

Surveillanceware prevention hinges on ensuring its kill chain remains incomplete. Blocking even one phase is sufficient. Given the distinct kill chains of mercenary spyware (see Section 5) and stalkerware (see Section 4), different approaches are necessary. For mercenary spyware, which exploits OS or application vulnerabilities, we need to improve our ability to quickly identify and patch these flaws. Conversely, for stalkerware, which abuses legitimate OS mechanisms, enhancements in the design and implementation of those mechanisms are crucial.

In the remainder of this section, we first examine software vulnerability identification and repair in mobile systems (Section 6.1.1). We discuss its open issues, review state-of-the-art research, and highlight promising avenues for future work. Next, shift our focus to the abuse of operating system mechanisms (Section 6.1.2), also discussing relevant state-of-the-art research, addressing its challenges, and pinpointing important directions for future investigation.

6.1.1. Improving Software Vulnerabilities Detection Methods

The examples of exploit chains we discussed in previous sections (Sections 5.1.1 and 5.1.2) demonstrate that mercenary spyware typically exploit both zero-day and one-day vulnerabilities. Hence, both types of vulnerabilities need to be patched as quickly as possible to reduce the window of opportunity for their successful exploitation by mercenary spyware. During the period elapsing from when the vulnerability has been identified to when the corresponding patch has been installed on a device, the smartphone remains indeed unprotected from the exploitation of that vulnerability.

The exploit chains discussed in previous sections (Section 5.1.1) demonstrate that mercenary spyware typically leverages both zero-day and one-day vulnerabilities. Therefore, patching both types of vulnerabilities as swiftly as possible is critical to reduce the window of opportunity for the successful exploitation by mercenary spyware. During the period between the identification of a vulnerability and the installation of the corresponding patch on a device, the smartphone remains unprotected from exploitation.

Addressing a zero-day vulnerability first necessitates its identification, followed by the development of a corresponding patch (effectively transforming it into a one-day vulnerability). Conversely, fixing a one-day vulnerability solely involves rolling out the relevant patch to mobile devices. As discussed in Section 5, both application-level and OS-level vulnerabilities can be jointly exploited within the same chain, requiring that both be suitably addressed. We will discuss these aspects separately.

Vulnerability Identification and Patching

Identifying software vulnerabilities, particularly within large codebases such as mobile operating systems, is a complex, error-prone, and time-consuming human endeavor. These same challenges extend to patching, as even one-day vulnerabilities are known to remain unpatched for extended periods [63,64]. Consequently, automated vulnerability discovery and repair methods have long been an active area of research [65–69].

To address these issues, further research in automatic vulnerability discovery and repair is essential. This research must focus on two key objectives: enhancing the ability to find increasingly complex and elusive vulnerabilities in the shortest possible time, and effectively patching them automatically. Achieving this will drastically reduce the number of unpatched vulnerabilities, thereby shrinking the attack surface for mercenary spyware.

Automating code vulnerability discovery has garnered significant attention in the recent literature, with deep learning (DL) techniques being extensively explored [70–79]. However, applying these techniques to mobile operating systems and applications remains largely unexamined, with only a few exceptions [78]. Consequently, it is currently unclear whether DL algorithms can effectively identify vulnerabilities in this specific domain. Therefore, investigating their effectiveness for vulnerability detection in mobile environments represents a crucial avenue for future research.

Conducting such a study hinges on access to large datasets of vulnerable mobile code samples, as deep learning (DL) algorithm effectiveness is highly dependent on training data quantity and quality. Unfortunately, publicly available datasets adequately representing mobile application and operating system vulnerabilities are currently lacking [80,81] or out-of-date [82]. Developing these datasets is therefore another critical area for future research.

Finally, DL techniques effectively detect one-day vulnerabilities, but currently struggle with zero-day vulnerabilities [83], even when integrated with large language models (LLMs) [84]. Therefore, extending these systems to handle zero-day threats effectively is a critical area for future research.

Developing automated methods to repair discovered vulnerabilities is another critical area for future research [68,69]. Promptly fixing vulnerabilities is indeed crucial for minimizing exploitation risks, but addressing the vast number of vulnerabilities across existing software systems often demands specialized expertise, which the current pool of experienced developers cannot meet [85]. Additionally, manual vulnerability resolution is time-consuming. For instance, it has been reported that it takes an average of 4.4 weeks to fix a vulnerability after it has been identified [86]. This creates an urgent demand for automated repair solutions.

While deep learning (DL) methods are being explored for automatic vulnerability repair [87–90], their effectiveness specifically for mobile vulnerability repair remains unclear, much like their application in automated detection. Therefore, exploring their efficacy in this domain is another crucial area for future research.

Furthermore, not all the vulnerabilities are actually exploitable. Repairing those that cannot be exploited is a waste of resources that delays the development of fixes for those which instead are exploitable. To address this issue, automatic methods able to quantify the exploitability of vulnerabilities are of great interest, and are an active area of research [91–93]. Further research is, however, needed to both improve the accuracy of these methods, especially for mobile operating systems code.

Not all security vulnerabilities can actually be exploited. Fixing those that cannot be exploited wastes valuable resources and delays the development of crucial patches for those that are truly dangerous. To tackle this, automatic methods for quantifying vulnerability and exploitability are drawing significant interest and are an active area of research [91–93].

However, more research is still needed to improve the accuracy of these methods, especially for mobile operating system code.

Rollout of Patches

Patch development is only one part of neutralizing the danger from software vulnerabilities. The other is ensuring that patches are actually installed on vulnerable devices. Both iOS and Android provide mechanisms to notify users about available security patches and new OS/application releases. However, since users are not forced to install updates once notified, the only control developers have is to roll out fixed vulnerability patches as quickly as possible.

Both Android and iOS strive to quickly fix operating system vulnerabilities and urge users to install updates. However, their differing approaches result in varying levels of patch rollout timeliness.

Apple does not disclose vulnerabilities it discovers until the corresponding patch is released [94]. Consequently, there is no publicly available data on how long it takes Apple to issue a patch after a vulnerability is identified. However, for zero-day vulnerabilities or one-day vulnerabilities known to be actively exploited in the wild, Apple releases urgent security updates with only a 1–3-day delay from their discovery [94].

The situation with Android is quite different due to its fragmented and often inconsistent ecosystem, which complicates OS updating and upgrading. This process involves numerous actors, including Google (the Android kernel source developer), device manufacturers, chip-set vendors, and frequently mobile carriers [95]. Consequently, Android security-critical kernel patches often lag significantly behind the mainstream Linux kernel, with over 20% experiencing a one-year delay [96]. Furthermore, the median latency for patch rollout has been measured at 24 days, with an additional median delay of 11 days before reaching the end device [95]. This process also shows high variance among different manufacturers [97], and older models may receive updates up to six months after their initial release [98].

To address Android fragmentation, Google has launched several key initiatives, including *Project Mainline* and the *Generic Kernel Image* [99]. These programs modularize and standardize the OS and kernel components, significantly reducing the burden on manufacturers and accelerating the delivery of security updates directly to users. These initiatives show promising results: a study of approximately 1000 smartphones from the ten largest Android manufacturers released between 2018 and 2023 demonstrated an attack prevention rate of about 85% with these efforts, significantly up from the 29–55% achieved by manufacturer-customized kernels alone [96].

Application vulnerabilities pose a significant threat, especially concerning surveillanceware attacks, as highlighted by recent reports [3,25,46,100]. Unfortunately, the burden of discovering and patching these vulnerabilities rests entirely with developers, who currently face no obligations or deadlines to implement fixes. To tackle this issue, app stores should expand their vetting processes to include automatic vulnerability scanning. Recent research [78] confirms the existence of effective automated techniques that could be used for this purpose. This scanning should be complemented by a mechanism that flags vulnerable applications, preventing their download until they are patched and blocking their use on users smartphones.

6.1.2. Strengthening Operating System Mechanisms

As mentioned in Section 4, stalkerware primarily leverages the abuse of legitimate operating system mechanisms. Therefore, effective prevention requires users to be keenly

aware of the permissions held by all installed applications, besides being vigilant about physical device security.

To support users in this complex task, operating systems should provide mechanisms that both simplify app permission monitoring and restrict granted permission types to the absolute minimum required for an app's intended behavior.

One effective approach to simplify permission monitoring is to provide users with a persistent, non-dismissible visual indicator whenever a granted permission is actively being used by any application. For instance, Android provides an indicator for location, screen recording, microphone, and camera usage [28]. Unfortunately the effectiveness of a persistent indicator can vary significantly for short-lasting actions (e.g., taking a screenshot) and can easily be missed if the phone is not actively being viewed (e.g., in a pocket or purse) [101]. Furthermore, persistent indicators can be masked by benign activity. For example, a microphone indicator active during a call will not differentiate between legitimate use and simultaneous background recording by stalkerware. Sophisticated surveillanceware can even time its actions to coincide with benign indicator triggers, thereby reducing suspicion and detectability.

A more comprehensive approach involves offering users a convenient way to review all permissions granted to installed applications, such as the Android *Privacy Dashboard* which showcases instances in which sensitive data like location, microphone, and camera have been accessed over the past 24 h [28]. While this dashboard represents a more holistic mechanism than real-time visual indicators, it inherently requires proactive user engagement. Therefore, to maximize its effectiveness, it should be extended to provide periodic notifications to users about applications with excessive permissions.

Regarding additional restrictions on app resource access, two main approaches have been explored [28]. One method involves mandating a specific app state for resource access. For instance, Android restricts clipboard access to only the default input method editor or the app currently in focus. Another strategy is to require apps to possess additional, often layered, permissions. Android exemplifies this by requiring an app to not only have audio recording permission but also function as an accessibility service to capture audio input during phone calls. However, a significant limitation of these permission-based restrictions emerges when an abuser has physical access to the device. In such scenarios, these restrictions often become ineffective, as the abuser can then arbitrarily grant permissions, bypassing the intended safeguards [28].

A more promising approach involves modeling the usage of potentially abusable services as pipelines of sandboxed, system-side code modules and meticulously policing their data flows [102]. This strategy enables more fine-grained control over how applications access and utilize the features provided by these services. Such a balance allows the operating system to preserve essential user functionality while significantly reducing privacy risks. While this approach has been demonstrated as feasible within the Android accessibility framework, further research is needed to prove its broader applicability to other types of abusable services. This approach, however, faces significant challenges. First, such defenses only work with repeated user approval, leading to considerable usability issues [28]. Furthermore, implementing these mechanisms without compromising user experience may require substantial smartphone resources.

6.2. Proposed Surveillanceware Detection Methods: Frontier and Open Challenges

Surveillanceware detection methods are essential because prevention measures cannot offer 100% coverage. As explored in Section 6.1.1, zero-day vulnerabilities are difficult to detect automatically, the number of undetected software vulnerabilities can be substantial [103,104], and their remediation may take significant time. Furthermore,

as discussed in Section 6.1.2, stalkerware prevention strategies rely on effective user cooperation. Consequently, users exhibiting poor cooperation face an increased risk of stalkerware compromise.

This means surveillanceware can bypass protective measures. Therefore, to safeguard users, it is crucial to detect its presence on a smartphone so that appropriate remedial actions, such as removal, can be undertaken.

Surveillanceware detection is a specialized subset of the broader malware detection problem, meaning general malware detection methods are, in principle, applicable. These methods typically involve collecting features from suspicious applications and comparing them against known malware patterns. Existing approaches can be broadly categorized as *signature-based*, *IoC-based*, *behavior-based*, and *Machine Learning/Deep Learning-based*.

6.2.1. Signature-Based Methods

Signature-based approaches compare a suspected app's unique byte sequence (its *signature*) against a database of known malware signatures. This approach is characteristic of malware scanning applications, which users commonly install on their smartphones for the real-time analysis of installed and executing applications.

While simple to implement and resource-efficient, these techniques perform poorly against surveillanceware [5,32,105,106] and malware in general [107,108]. They particularly struggle with polymorphic, metamorphic, obfuscated, and dynamically loaded malware, as crucial "true features" might be hidden or only revealed during runtime [52].

6.2.2. IoC-Based Methods

Indicators of Compromise (IoC)-based approaches offer an improvement over traditional signature-based methods. These approaches leverage data left by surveillanceware on a device during its execution, enabling not only detection but also attribution to a specific adversary. Simple indicators (e.g., app package name, installation date, requested permissions) have proven effective for stalkerware detection [30,106]. This approach is employed by some forensics tools, such as Warne [30]. However, it proves ineffective against mercenary spyware, which typically does not require installation and instead necessitates analyzing data generated during its execution, as demonstrated by the Mobile Verification Toolkit (MVT) [109].

Despite being an improvement over signature-based methods, IoC-based approaches suffer from various problems. Their user-dependent efficacy means they can fail if users lack sufficient diligence to use them. Furthermore, IoCs are susceptible to changes in surveillanceware behavior, making them less reliable. Finally, these techniques often rely on privilege-restricted data that is inaccessible or prone to alteration without superuser privileges [52], necessitating complex technical extraction procedures from the device [109].

6.2.3. Behavior-Based Methods

Behavior-based approaches [110] monitor the dynamic behavior (e.g., system calls, network traffic, etc.) of applications on a device, looking for patterns or anomalies that may indicate the presence of surveillanceware. Although they offer a detection rate better than previous approaches, they still perform unsatisfactorily for the following reasons [13,111]. First, surveillanceware employs sophisticated stealthiness and evasion tactics [27]), making its true malicious functionality hard to observe. Second, surveillanceware often exhibit behavioral similarity to benign applications [112] or have dual-use functionalities [13,32] (e.g., anti-theft or child safety features), which complicates distinguishing them from legitimate apps. Third, the lack of prior knowledge about previously unseen surveillanceware poses a significant challenge, as there is no baseline for comparison [113–116]. Finally, surveillanceware leverages anti-characterization techniques to avoid full execution or re-

veal its complete functionality within controlled environments like virtual machines or sandboxes [52,105,110].

6.2.4. Machine Learning-Based Methods

Machine Learning (ML)-based and *Deep Learning (DL)-based* approaches operate by collecting data, extracting features, training a model, and then using that model for classification [107,114,115,117–119]. While these methods demonstrate superior detection performance compared to traditional behavior-based techniques, notably in handling obfuscated/encrypted/packed code and previously unseen malware, they still face several persistent challenges that necessitate further research. These challenges include

- *Generalization vs. specificity*: much current research prioritizes general malware detection rather than focusing on the unique, stealthy characteristics of surveillanceware [13]. This broad approach presents a challenge: legitimate monitoring applications can generate network traffic that closely resembles surveillanceware, potentially leading to false positives [111]). Therefore, future research needs to concentrate on developing methods specifically tailored for surveillanceware detection, moving beyond a general malware detection focus;
- *Data quality and ambiguity*: Developing robust behavioral models for detecting threats like surveillanceware is tough because it requires high-quality, labeled datasets. These datasets are both expensive and time-consuming to create. Real-world surveillanceware samples are rare, which often leads to skewed training data. Plus, inconsistent labels or conflicting results from different antivirus engines make it hard to establish a reliable ground truth. While recent literature shows ongoing work on dataset construction [111,120,121], more research is needed to develop methodologies for generating suitable datasets. To combat the scarcity of malware samples, training methods that need less data, like Few- shot learning [122], have been explored [114]. However, further research is still required to fully assess their performance when specifically dealing with surveillanceware;
- *Challenges with zero-day malware*: ML/DL methods still struggle to effectively detect previously unseen malware—malware unrelated to their training data [113,114]. Therefore, further research is crucial to either improve existing ML/DL methods or explore novel approaches that enhance their ability to detect new and evolving malware threats;
- *Feature engineering challenges*: Identifying behavioral characteristics uniquely indicative of specific malware families proves difficult, as behaviors can vary across families yet share similarities within them [13,123]. For example, broad permission requests may not accurately reflect malicious intent, given that many benign apps require similar permissions;
- *Lack of transparency*: Many DL models are considered "black-box" models because they often cannot provide clear, human-interpretable explanations for classifying a specific application as malicious [115]. This lack of transparency can hinder trust for security analysts who need to understand the underlying causality of a detection [118];
- *Model aging and evolution*: The performance of ML/DL models can degrade significantly over time due largely to the rapid evolution of mobile surveillanceware and the emergence of new variants and families. This necessitates the continuous retraining and adaptation of detection models [117].

7. Discussion

Protecting users from surveillanceware is a top priority for mobile operating system manufacturers. Both Google and Apple consistently enhance their security mechanisms to achieve this.

Google's real-time mobile malware protection on Android primarily relies on *Google Play Protect*, which scans billions of apps daily using a hybrid on-device and cloud-based analysis for suspicious behaviors and potentially harmful applications. Defenses are further strengthened by collaborative initiatives like the *App Defense Alliance* and *Google Project Zero*, focusing on threat intelligence sharing and zero-day vulnerability disclosure. For high-risk users, Android 16's *Advanced Protection* provides Google's strongest security measures, including features for forensic analysis.

Apple's real-time mobile malware protection on iOS leverages its tightly integrated hardware and software ecosystem. Foundational security is rooted in the *Secure Enclave* and custom *Apple Silicon*, enabling features like *Kernel Integrity Protection* and *Pointer Authentication Codes* to prevent exploitation. Proactive defenses such as *BlastDoor*, *Apple Threat Notifications*, and *Lockdown Mode*, along with the *Endpoint Security Framework*, collectively work to detect and mitigate sophisticated attacks.

Despite these protective measures, our comprehensive analysis of surveillanceware capabilities and the state of the art of countermeasures indicates that, despite the ongoing efforts of Google and Apple, both Android and iOS struggle to protect devices and users, especially against sophisticated mercenary spyware attacks. While iOS is generally considered more secure due to its architectural design and robust mechanisms [34,35], both platforms remain vulnerable to these sophisticated threats.

Therefore, further research is needed in order to properly address the research challenges that emerge from our analysis, and in particular:

- *Improve the methods to automatically identify and repair software vulnerabilities*: A key challenge lies in proactively and rapidly identifying complex, elusive mobile vulnerabilities. This effort is hampered by the unproven effectiveness of current DL algorithms in this domain, also caused by the lack of adequate public datasets, and their ongoing struggle with zero-day vulnerabilities. Therefore, future research must focus on investigating DL efficacy for mobile vulnerability detection, developing robust datasets, extending DL with large language models (LLMs) to enhance zero-days vulnerability detection, and creating automated vulnerability repair methods for mobile platforms;
- *Strengthen operating system mechanisms*: The challenge lies in the varying effectiveness of persistent visual indicators and the reactive nature of privacy dashboards. These issues are compounded by permission-based restrictions being easily bypassed with physical access. Future research should therefore concentrate on extending privacy dashboards with proactive notifications and exploring sandboxed, system-side module pipelines to achieve more fine-grained control over abusable services;
- *Improve surveillanceware detection methods*: Traditional malware detection methods struggle against sophisticated surveillanceware due to its stealthiness, evasive tactics, and behavioral similarities to benign applications. While ML/DL models offer promising advancements, they face additional challenges requiring further research. These include issues with the specificity of detection techniques, dataset quality and acquisition, difficulty with previously unseen malware, feature engineering complexities, lack of transparency, and model aging.

A common difficulty across all the research activities listed above is the lack of source code availability for operating systems and applications. iOS, for instance, is closed-source. It is also misleading to assume that the Android's open source nature negates this issue; device manufacturers' customizations can, and typically do, include proprietary software modules for which source code is not available. The same applies to applications. Therefore, making the source code of these systems and applications accessible would be an essential contribution, enabling the development of truly effective protection mechanisms capable of keeping pace with surveillanceware's evolution.

8. Conclusions

In this paper, we thoroughly reviewed mobile surveillanceware, classifying it as either stalkerware (abusive, often requiring physical access) or mercenary spyware (used for espionage, employing sophisticated remote exploits). We detailed their extensive data collection capabilities and how they bypass mobile OS protections (like sandboxing and access control) due to risky user behaviors and exploitable vulnerabilities. Finally, we discussed countermeasures, focusing on technical prevention and detection strategies rather than avoidance.

Our analysis shows that, despite the efforts by operating systems and device manufacturers to counter spyware, successful attacks continue to occur. This ongoing challenge necessitates further research to enhance defenses.

In particular, while significant strides have been made in understanding and combating mobile spyware, the persistent and evolving arms race between attackers and defenders necessitates a sustained and targeted research effort. This critical work must span several key domains: enhancing the identification and remediation of software vulnerabilities across mobile operating systems and applications, hardening legitimate operating system mechanisms that are currently exploited by spyware, and developing advanced detection techniques. Crucially, this future research must increasingly leverage the power of artificial intelligence, including deep learning and large language models, to keep pace with and ultimately overcome the sophisticated tactics employed by modern spyware.

While technical countermeasures are essential, they alone are insufficient to fully combat mobile surveillance, as even the most sophisticated and effective technical solutions can never guarantee perfect protection. Effective defense thus demands robust political responses, specifically through well-crafted policies and regulations (e.g., [124,125]). These measures are crucial for establishing clear legal frameworks that both define and prohibit the misuse of surveillance technologies. By creating strong disincentives and ensuring accountability, appropriate political action can significantly mitigate the pervasive threat to privacy posed by mobile surveillanceware, complementing technical efforts with essential legal and ethical safeguards.

Finally, beyond technical and political safeguards, user awareness and responsible behavior are crucial. Avoiding risky online activities and app installations significantly reduces the attack surface for surveillanceware. Ultimately, vigilant user practices form a vital line of defense, complementing other countermeasures to keep devices and data secure.

Funding: This paper is part of the project NODES, which has received funding from the MUR-M4C2 1.5 of PNRR funded by the European Union-NextGenerationEU (Grant agreement no. ECS00000036).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kumar, A.; Del Rosso, K.; Albrecht, J.; Hebeisen, C. *Mobile APT Surveiilance Campaigns Targeting Uyghurs*; Technical Report; Lookout Inc.: San Francisco, CA, USA, 2020.
2. Stafford, T.F.; Urbaczewski, A. Spyware: The ghost in the machine. *Commun. Assoc. Inf. Syst.* **2004**, *14*. [CrossRef]
3. Huntley, S. *Buying Spying: How the Commercial Surveillance Industry Works and What Can Be Done About It*; Technical Report; Google Threat Analysis Group: Boston, MA, USA, 2024.
4. Harkin, D.; Molnar, A.; Vowles, E. The commodification of mobile phone surveillance: An analysis of the consumer spyware industry. *Crime Media Cult. Int. J.* **2020**, *16*, 33–60. [CrossRef]
5. Chatterjee, R.; Doerfler, P.; Orgad, H.; Havron, S.; Palmer, J.; Freed, D.; Levy, K.; Dell, N.; McCoy, D.; Ristenpart, T. The Spyware Used in Intimate Partner Violence. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20–24 May 2018.

6. Freed, D.; Havron, S.; Tseng, E.; Gallardo, A.; Chatterjee, R.; Ristenpart, T.; Dell, N. "Is my phone hacked?" Analyzing Clinical Computer Security Interventions with Survivors of Intimate Partner Violence. *Proc. ACM Hum.-Comput. Interact.* **2019**, *3*, 1–24. [CrossRef]
7. Gibson, G.; Frost, V.; Platt, K.; Garcia, W.; Vargas, L.; Rampazzi, S.; Bindschaedler, V.; Traynor, P.; Butler, K. Analyzing the Monetization Ecosystem of Stalkerware. *Proc. Priv. Enhancing Technol.* **2022**, *2022*, 105–119. [CrossRef]
8. Roundy, K.A.; Mendelberg, P.B.; Dell, N.; McCoy, D.; Nissani, D.; Ristenpart, T.; Tamersoy, A. The Many Kinds of Creepware Used for Interpersonal Attacks. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 18–21 May 2020.
9. Almansoori, M.; Gallardo, A.; Poveda, J.; Ahmed, A.; Chatterjee, R. A Global Survey of Android Dual-Use Applications Used in Intimate Partner Surveillance. *Proc. Priv. Enhancing Technol.* **2022**, *2022*, 120–139. [CrossRef]
10. Chourasiya, S.; Samanta, G.; Sardar, D.K.; Sharma, P.; Kumar, C.V. Pegasus Spyware: A Vulnerable Behaviour-based Attack System. In Proceedings of the 2nd International Conference on Edge Computing and Applications, Namakkal, India, 19–21 July 2023.
11. Rudie, J.; Katz, Z.; Kuhbander, S.; Bhunia, S. Technical Analysis of the NSO Group's Pegasus Spyware. In Proceedings of the International Conference on Computational Science and Computational Intelligence, Las Vegas, NV, USA, 15–17 December 2021.
12. Karwan, M.K. A comphrensive analysis of Pegasus spyware and its implications for digital privacy and security. *Int. J. Intell. Syst. Appl. Eng.* **2024**, *12*, 1360–1373.
13. Naser, M.; Albazar, H.; Abdel-Jaber, H. Mobile spyware identification and categorization: A systematic review. *Informatica* **2023**, *47*, 45–56. [CrossRef]
14. Hayes, D.; Cappa, F.; Le-Khac, N.A. An effective approach to mobile device management: Security and privacy issues associated with mobile applications. *Digit. Bus.* **2020**, *1*, 1. [CrossRef]
15. Delgado-Santos, P.; Stragapede, G.; Tolosana, R.; Guest, R.; Deravi, F.; Vera-Rodriguez, R. A Survey of Privacy Vulnerabilities of Mobile Device Sensors. *ACM Comput. Surv.* **2022**, *54*, 1–30. [CrossRef]
16. Boussada, R.; Bouchaala, M.; Saidane, L.A. Privacy and Tracking in the Emerging Mobile Applications: A Survey. In Proceedings of the International Wireless Communications and Mobile Computing, Marrakesh, Morocco, 19–23 June 2023.
17. Patil, H.; Sharma, K. Assessing the Landscape of Mobile Data Vulnerabilities: A Comprehensive Review. In Proceedings of the International Conference on Computational Intelligence and Sustainable Engineering Solutions, Marrakesh, Morocco, 19–23 June 2023.
18. The State of Stalkerware in 2023. Available online: https://securelist.com/state-of-stalkerware-2023/112135/ (accessed on 30 June 2025).
19. Kyle Hiebert. The Growing Global Spyware Industry Must Be Reined in. Available online: https://www.cigionline.org/articles/the-growing-global-spyware-industry-must-be-reined-in/ (accessed on 30 June 2025).
20. Chin-Rothmann, C. Cyber Mercenaries: Limiting Government Use of Commercial Spyware. Georget. J. Int. Aff. 2024. Available online: https://gjia.georgetown.edu/2024/09/04/cyber-mercenaries-limiting-government-use-of-commercial-spyware/ (accessed on 30 June 2025).
21. Bintang Timur, F. *Cyber Mercenaries: The Failures of Current Responses and the Imperative of International Collaboration*; Technical Report; The Observer Reseach Foundation: New Delhi, India, 2023.
22. Amnesty International Security Lab. Predator Files: Technical Deep-Dive into Intellexa Alliance's Surveillance Products. Available online: https://securitylab.amnesty.org/latest/2023/10/technical-deep-dive-into-intellexa-alliance-surveillance-products/ (accessed on 29 April 2025).
23. Mobile Operating System Market Share Worldwide-June 2025. Available online: https://gs.statcounter.com/os-market-share/mobile/worldwide (accessed on 30 June 2025).
24. Mayrhofer, R.; Stoep, J.V.; Brubaker, C.; Kralevich, N. The Android Platform Security Model. *ACM Trans. Priv. Secur.* **2021**, *24*, 19. [CrossRef]
25. Rahkema, K.; Pfahl, D. Quality Analysis of iOS Applications with Focus on Maintainability and Security. In Proceedings of the 38th IEEE International Conference on Software Maintenance and Evolution, Limassol, Cyprus, 3–7 October 2022.
26. Common Vulnerabilities and Exposures (CVE) Database. Available online: https://www.cve.org/ (accessed on 5 May 2025).
27. Graf, K.; Lerga, J.; Dobraš, B. Data Collection and Hiding Capabilities of Android Stalkerware. In Proceedings of the IEEE 21st Jubilee International Symposium on Intelligent Systems and Informatics, Pula, Croatia, 21–23 September 2023.
28. Liu, E.; Rao, S.; Havron, S.; Ho, G.; Savage, S.; Voelker, G.M.; McCoy, D. No privacy among spies: Assessing the functionality and insecurity of consumer android spyware apps. *Proc. Priv. Enhancing Technol.* **2023**, *2023*, 207–224. [CrossRef]
29. Mangeard, P.; Yu, X.; Mannan, M.; Youssef, A. No Place to Hide: Privacy Exposure in Anti-stalkerware Apps and Support Websites. In *Secure IT Systems*; Spinger: Berlin/Heidelberg, Germany, 2024.
30. Mangeard, P.; Tejaswi, B.; Mannan, M.; Youssef, A. WARNE: A stalkerware evidence collection tool. *Forensic Sci. Int. Digit. Investig.* **2024**, *48*, 301677. [CrossRef]
31. Baraniuk, C. The rise of stalkerware. *New Sci.* **2019**, *244*, 20–21. [CrossRef]

32. Bonam, M.; Rayavaram, P.; Abbasalizadeh, M.; Lee, C.; Pattavina, A.; Narain, S. Current Research, Challenges, and Future Directions in Stalkerware Detection Techniques for Mobile Ecosystems. In Proceedings of the 11th International Conference on Information Systems Security and Privacy, Porto, Portugal, 20–22 February 2025.
33. Suau, R. Analysis of a Malware Exploiting Android Accessibility Services. Available online: https://blog.pradeo.com/accessibility-services-mobile-analysis-malware (accessed on 22 April 2025).
34. Garg, S.; Baliyan, N. Comparative analysis of Android and iOS from security viewpoint. *Comput. Sci. Rev.* **2021**, *40*, 100372. [CrossRef]
35. Harkin, D.; Molnar, A. Operating-System Design and Its Implications for Victims of Family Violence: The Comparative Threat of Smart Phone Spyware for Android Versus iPhone Users. *Violence Women* **2021**, *27*, 851–875. [CrossRef]
36. Talos, C. Mercenary Mayhem: A Technical Analysis of Intellexa's Predator Spyware. Available online: https://blog.talosintelligence.com/mercenary-intellexa-predator/ (accessed on 29 April 2025).
37. Bazaliy, M.; Flossman, M.; Blaich, A.; Hardy, S.; Edwards, K.; Murray, M. *Technical Analysis of Pegasus Spyware: An Investigation Into Highly Sophisticated Espionage Software*; Technical Report; Lookout Inc.: San Francisco, CA, USA, 2016.
38. Bazaliy, M.; Neckar, C.; Sinclair, G.; in7egral. *Techical Analysis of the Pegasus Exploits on iOS*; Technical Report; Lookout Inc.: San Francisco, CA, USA, 2016.
39. CyberMasterV. A Technical Analysis of Pegasus for Android—Part 1. Available online: https://cybergeeks.tech/a-technical-analysis-of-pegasus-for-android-part-1/ (accessed on 21 May 2025).
40. CyberMasterV. A Technical Analysis of Pegasus for Android—Part 2. Available online: https://cybergeeks.tech/a-technical-analysis-of-pegasus-for-android-part-2/ (accessed on 21 May 2025).
41. CyberMasterV. A Technical Analysis of Pegasus for Android—Part 3. Available online: https://cybergeeks.tech/a-technical-analysis-of-pegasus-for-android-part-3/ (accessed on 21 May 2025).
42. Intellexa Limited. Jupiter: Network 0-Click Solution in HTTPS Traffic. Available online: https://www.woz.ch/files/text/2023/produktbroschuere_jupiter.pdf (accessed on 21 May 2025).
43. Intellexa Limited. Mars: ISP Interception. Available online: https://www.woz.ch/files/text/2023/produktbroschuere_mars.pdf (accessed on 21 May 2025).
44. Intellexa Limited. Triton: Innovative Tactical Cyber Solution. Available online: https://www.woz.ch/files/text/2023/produktbroschuere_triton.pdf (accessed on 21 May 2025).
45. Pegasus-Product Description. Available online: https://ia801005.us.archive.org/1/items/nso-pegasus/NSO-Pegasus.pdf (accessed on 21 May 2025).
46. Marczak, B.; Scott-Railton, J.; Robertson, K.; Perry, A.; Brown, R.; Razzak, B.A.; Anstis, S.; Deibert, R. *Virtue or Vice? A First Look at Paragon's Proliferating Spyware Operations*; Technical Report 183; University of Toronto: Toronto, ON, Canada, 2025.
47. Everything You Need to Know About the Pegasus Spyware. Available online: https://www.cloudsek.com/blog/everything-you-need-to-know-about-the-pegasus-spyware (accessed on 27 April 2025).
48. Clement Lecign. Spyware Vendors Use 0-Days and n-Days Against Popular Platforms. Available online: https://blog.google/threat-analysis-group/spyware-vendors-use-0-days-and-n-days-against-popular-platforms/ (accessed on 29 April 2025).
49. Kaspersky Team. TriangleDB: The Spyware Implant of Operation Triangulation. Available online: https://www.kaspersky.com/blog/triangledb-mobile-apt/48471/ (accessed on 29 May 2025).
50. Operation Triangulation. Available online: https://securelist.com/trng-2023/ (accessed on 29 May 2025).
51. Boris Larin. Operation Triangulation: The Last (Hardware) Mystery. Available online: https://securelist.com/operation-triangulation-the-last-hardware-mystery/111669/ (accessed on 29 May 2025).
52. Javaheri, D.; Hosseinzadeh, M.; Rahmani, A.M. Detection and Elimination of Spyware and Ransomware by Intercepting Kernel-Level System Routines. *IEEE Access* **2018**, *6*, 78321–78332. [CrossRef]
53. Marczak, B.B.; Scott-Railton, J.; Razzak, B.A.; Aljizawi, N.; Anstis, S.; Berdan, K.; Deibert, R. *Pegasus vs. Predator: Dissident's Doubly-Infected iPhone Reveals Cytrox Mercenary Spyware*; Technical Report 147; University of Toronto: Toronto, ON, Canada, 2021.
54. Lookout Inc. Predator & Pegasus. Available online: https://www.lookout.com/threat-intelligence/article/predator-pegasus (accessed on 25 May 2025).
55. Whittaker, Z. Spyzie Stalkerware is Spying on Thousands of Android and iPhone Users. Available online: https://techcrunch.com/2025/02/27/spyzie-stalkerware-spying-on-thousands-of-android-and-iphone-users/ (accessed on 9 June 2025).
56. Lorenzo Franceschi-Bicchierai. Hacked, Leaked, Exposed: Why You Should Never Use Stalkerware Apps. Available online: https://techcrunch.com/2025/03/19/hacked-leaked-exposed-why-you-should-stop-using-stalkerware-apps/ (accessed on 9 June 2025).
57. Whittaker, Z. Stalkerware Apps Cocospy and Spyic Are Exposing Phone Data of Millions of People. Available online: https://techcrunch.com/2025/02/20/stalkerware-apps-cocospy-spyic-exposing-phone-data-of-millions-of-people/ (accessed on 9 June 2025).

58. Amnesty International. Global Ruling Against NSO Group in Whatsapp Case a "Momentous Win in Fight Against Spyware Abuse". Available online: https://www.amnesty.org/en/latest/news/2025/05/ruling-against-nso-group-in-whatsapp-case-a-momentous-win/ (accessed on 9 June 2025).
59. Amnesty International. Europe: Paragon Attacks Highlight Europe's Growing Spyware Crisis. Available online: https://www.amnesty.org/en/latest/news/2025/03/europe-paragon-attacks-highlight-europes-growing-spyware-crisis/ (accessed on 9 June 2025).
60. Chin, E.; Felt, A.P.; Sekar, V.; Wagner, D. Measuring user confidence in smartphone security and privacy. In Proceedings of the Eighth Symposium on Usable Privacy and Security, Washington, DC, USA, 11–13 July 2012.
61. Butler, R. A systematic literature review of the factors affecting smartphone user threat avoidance behaviour. *Inf. Comput. Secur.* **2020**, *28*, 555–574. [CrossRef]
62. Dawie, F.J.; Masrek, M.N.; Rahman, S.A. Systematic Literature Review: Information security behaviour on smartphone users. *Environ.-Behav. Proc. J.* **2022**, *7*, 275–281. [CrossRef]
63. Hooimeijer, P.; Weimer, W. Modeling bug report quality. In Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, Atlanta, GA, USA, 5–9 November 2007.
64. Ding, Z.Y.; Goues, C.L. An Empirical Study of OSS-Fuzz Bugs. In Proceedings of the 18th IEEE/ACM International Conference on Mining Software Repositories, Madrid, Spain, 17–19 May 2021.
65. Pham, N.H.; Nguyen, T.T.; Nguyen, H.A.; Nguyen, T.N. Detection of recurring software vulnerabilities. In Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 20–24 September 2010.
66. Shin, Y.; Meneely, A.; Williams, L.; Osborne, J.A. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Trans. Softw. Eng.* **2011**, *37*, 772–787. [CrossRef]
67. Ghaffarian, S.M.; Shahriari, H.R. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. *ACM Comput. Surv.* **2017**, *50*, 197158–197172. [CrossRef]
68. Le Goues, C.; Pradel, M.; Roychoudhury, A. Automated program repair. *Commun. ACM* **2019**, *62*, 56–65. [CrossRef]
69. Shariffdeen, R.; Noller, Y.; Grunske, L.; Roychoudhury, A. Concolic program repair. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, 20–25 June 2021.
70. Chakraborty, S.; Krishna, R.; Ding, Y.; Ray, B. Deep Learning Based Vulnerability Detection: Are We There Yet? *IEEE Trans. Softw. Eng.* **2022**, *48*, 3280–3296. [CrossRef]
71. Chakraborty, P.; Arumugam, K.K.; Alfadel, M.; Nagappan, M.; McIntosh, S. Revisiting the Performance of Deep Learning-Based Vulnerability Detection on Realistic Datasets. *IEEE Trans. Softw. Eng.* **2024**, *50*, 2163–2177. [CrossRef]
72. Mahbub, M.; Khan, M.S.A.; Hamid, T.; Mia, M.S. A Novel Vulnerability Exploit Prediction System Using the Relational Vulnerability-Vendor Network. *Digit. Threat.* **2025**, *6*, 1–17. [CrossRef]
73. Marjanov, T.; Pashchenko, I.; Massacci, F. Machine Learning for Source Code Vulnerability Detection: What Works and What Isn't There Yet. *IEEE Secur. Priv.* **2022**, *20*, 60–76. [CrossRef]
74. Sejfia, A.; Das, S.; Shafiq, S.; Medvidović, N. Toward Improved Deep Learning-based Vulnerability Detection. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, Lisbon, Portugal, 14–20 April 2024.
75. Cotroneo, D.; Grasso, F.C.; Natella, R.; Orbinato, V. Can Neural Decompilation Assist Vulnerability Prediction on Binary Code? In Proceedings of the 18th European Workshop on Systems Security, Rotterdam, The Netherlands, 30 March–3 April 2025.
76. Harzevili, N.S.; Belle, A.B.; Wang, J.; Wang, S.; Jiang, Z.M.J.; Nagappan, N. A Systematic Literature Review on Automated Software Vulnerability Detection Using Machine Learning. *ACM Comput. Surv.* **2025**, *57*, 1–36. [CrossRef]
77. Steenhoek, B.; Rahman, M.M.; Jiles, R.; Le, W. An Empirical Study of Deep Learning Models for Vulnerability Detection. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering, Melbourne, Australia, 14–20 May 2023.
78. Senanayake, J.; Kalutarage, H.; Al-Kadri, M.O.; Petrovski, A.; Piras, L. Android Source Code Vulnerability Detection: A Systematic Literature Review. *ACM Comput. Surv.* **2023**, *55*, 187. [CrossRef]
79. Mathews, N.S.; Brus, Y.; Aafer, Y.; Nagappan, M.; McIntosh, S. LLbezpeky: Leveraging Large Language Models for Vulnerability Detection. *arXiv* **2024**, arXiv:2401.0126.
80. Renjith, G.; Aji, S. Unveiling the Security Vulnerabilities in Android Operating System. In Proceedings of the 2nd International Conference on Sustainable Expert Systems, Hotel Himalaya Lalitpur, Nepal, 9–10 September 2022.
81. Bhurtel, M.; Rawat, D.B. Unveiling the Landscape of Operating System Vulnerabilities. *Future Internet* **2023**, *15*, 248. [CrossRef]
82. Challande, A.; David, R.; Renault, G. Building a Commit-level Dataset of Real-world Vulnerabilities. In Proceedings of the 12th ACM Conference on Data and Application Security and Privacy, Baltimore, MD, USA, 25–27 April 2022.
83. Lu, G.; Ju, X.; Chen, X.; Pei, W.; Cai, Z. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. *J. Syst. Softw.* **2024**, *212*, 112031. [CrossRef]
84. Ferrag, M.A.; Battah, A.; Tihanyi, N.; Jain, R.; Maimuţ, D.; Alwahedi, F.; Lestable, T.; Thandi, N.S.; Mechri, A.; Debbah, M.; et al. SecureFalcon: Are We There Yet in Automated Software Vulnerability Detection with LLMs? *IEEE Trans. Softw. Eng.* **2025**, *51*, 1248–1265. [CrossRef]

85. Ji, T.; Wu, Y.; Wang, C.; Zhang, X.; Wang, Z. The Coming Era of AlphaHacking?: A Survey of Automatic Software Vulnerability Detection, Exploitation and Patching Techniques. In Proceedings of the IEEE 3rd International Conference on Data Science in Cyberspace, Guangzhou, China, 18–21 June 2018.
86. Forsgren, N.; Alberts, B.; Backhouse, K.; Baker, G.; Cecarelli, G.; Jedamski, D.; Kelly, S.; Sullivan, C. 2020 State of the Octoverse: Securing the World's Software. *arXiv* **2021**, arXiv:2110.10246.
87. Shariffdeen, R.; Timperley, C.S.; Noller, Y.; Le Goues, C.; Roychoudhury, A. Vulnerability Repair via Concolic Execution and Code Mutations. *ACM Trans. Softw. Eng. Methodol.* **2025**, *34*, 1–27. [CrossRef]
88. Zhou, X.; Kim, K.; Xu, B.; Han, D.; Lo, D. Out of Sight, Out of Mind: Better Automatic Vulnerability Repair by Broadening Input Ranges and Sources. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, Lisbon, Portugal, 14–20 April 2024.
89. Zhou, X.; Cao, S.; Sun, X.; Lo, D. Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* **2025**, *34*, 1–31. [CrossRef]
90. Wang, X.; Tian, Y.; Huang, K.; Liang, B. Practically implementing an LLM-supported collaborative vulnerability remediation process: A team-based approach. *Comput. Secur.* **2025**, *148*, 104113. [CrossRef]
91. Bhatt, N.; Anand, A.; Yadavalli, V.S.S. Exploitability prediction of software vulnerabilities. *Qual. Reliab. Eng. Int.* **2021**, *37*, 648–663. [CrossRef]
92. Tang, X.; Zhou, H.; Zhang, M.; Zhang, Y.; Wu, G.; Lu, H.; Yu, X.; Tian, Z. Research on the Exploitability of Binary Software Vulnerabilities. In Proceedings of the IEEE 12th International Conference on Cloud Networking, Hoboken, NJ, USA, 1–3 November 2023.
93. Iannone, E.; Sellitto, G.; Iaccarino, E.; Ferrucci, F.; Lucia, A.D.; Palomba, F. Early and Realistic Exploitability Prediction of Just-Disclosed Software Vulnerabilities: How Reliable Can It Be? *ACM Trans. Softw. Eng. Methodol.* **2024**, *33*, 1–41. [CrossRef]
94. Apple Inc. Apple Security Releases. Available online: https://support.apple.com/en-us/100100 (accessed on 12 April 2025).
95. Jones, K.R.; Yen, T.F.; Sundaramurthy, S.C.; Bardas, A.G. Deploying Android Security Updates: An Extensive Study Involving Manufacturers, Carriers, and End Users. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, 9–13 November 2020.
96. Maar, L.; Draschbacher, F.; Lamster, L.; Mangard, S. Defects-in-depth: Analyzing the integration of effective defenses against one-day exploits in android kernels. In Proceedings of the 33rd USENIX Conference on Security Symposium, Philadelphia, PA, USA, 14–16 August 2024.
97. Leierzopf, E.; Mayrhofer, R.; Roland, M.; Studier, W.; Dean, L.; Seiffert, M.; Putz, F.; Becker, L.; Thomas, D.R. A Data-Driven Evaluation of the Current Security State of Android Devices. In Proceedings of the IEEE Conference on Communications and Network Security, Taipei City, Taiwan, 30 September–3 October 2024.
98. Kumar, A.; Peck, M. Research Analysis and Guidance: Ensuring Android Security Update Adoption. Available online: https://techcommunity.microsoft.com/blog/vulnerability-management/research-analysis-and-guidance-ensuring-android-security-update-adoption/4216714 (accessed on 29 April 2005).
99. Google. Generic Kernel Image (GKI) Project. Available online: https://source.android.com/docs/core/architecture/kernel/generic-kernel-image (accessed on 14 June 2025).
100. Gao, J.; Li, L.; Kong, P.; Bissyande, T.F.; Klein, J. Understanding the Evolution of Android App Vulnerabilities. *IEEE Trans. Reliab.* **2021**, *70*, 212–230. [CrossRef]
101. Choe, Y.; Yu, H.; Kim, T.; Lee, S.; Lee, H.; Kim, H. (In)visible Privacy Indicator: Security Analysis of Privacy Indicator on Android Devices. In Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, Singapore, 1–5 July 2024.
102. Huang, J.; Backes, M.; Bugiel, S. A11y and Privacy do not have to be mutually exclusive: Constraining Accessibility Service Misuse on Android. In Proceedings of the 30th USENIX Security Symposium, Virtual Event, 11–13 August 2021.
103. Alhazmi, O.; Malaiya, Y.; Ray, I. Measuring, analyzing and predicting security vulnerabilities in software systems. *Comput. Secur.* **2007**, *26*, 219–228. [CrossRef]
104. Bhatt, N.; Anand, A.; Yadavalli, V.S.S.; Kumar, V. Modeling and Characterizing Software Vulnerabilities. *Int. J. Math. Eng. Manag. Sci.* **2017**, *2*, 288–299. [CrossRef]
105. Fassl, M.; Anell, S.; Houy, S.; Lindorfer, M.; Krombholz, K. Comparing User Perceptions of Anti-Stalkerware Apps with the Technical Reality. In Proceedings of the 18th Symposium on Usable Privacy and Security, Boston, MA, USA, 7–9 August 2022.
106. Han, Y.; Roundy, K.A.; Tamersoy, A. Towards Stalkerware Detection with Precise Warnings. In Proceedings of the 37th Annual Computer Security Applications Conference, Virtual Event, 6–10 December 2021.
107. Dahiya, A.; Sukhdip Singh, G.S. Android malware analysis and detection: A systematic review. *Expert Syst.* **2025**, *42*, e13488. [CrossRef]
108. Zhang, D.; Wu, X.; He, E.; Guo, X.; Yang, X.; Li, R.; Li, H. Android Malware Detection Based on Hypergraph Neural Networks. *Appl. Sci.* **2023**, *13*, 12629. [CrossRef]

109. Amnesty International. The Mobile Verification Toolkit. Available online: https://docs.mvt.re/en/latest (accessed on 10 March 2025).
110. EyalSalman, R.T. Android Stalkerware Detection Techniques: A Survey Study. In Proceedings of the IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology, Amman, Jordan, 22–24 May 2023.
111. Qabalin, M.K.; Naser, M.; Alkasassbeh, M. Android spyware detection using machine learning: A novel dataset. *Sensors* **2022**, *22*, 5765. [CrossRef] [PubMed]
112. Conti, M.; Rigoni, G.; Toffalini, F. ASAINT: A spy App identification system based on network traffic. In Proceedings of the 15th International Conference on Availability, Reliability and Security, Virtual Event, Ireland, 25–28 August 2020.
113. Abri, F.; Siami-Namini, S.; Khanghah, M.A.; Soltani, F.M.; Namin, A.S. Can Machine/Deep Learning Classifiers Detect Zero-Day Malware with High Accuracy? In Proceedings of the IEEE International Conference on Big Data, Los Angeles, CA, USA, 9–12 December 2019.
114. Fatemeh Deldar, M.A. Deep Learning for Zero-day Malware Detection and Classification: A Survey. *ACM Comput. Surv.* **2024**, *56*, 1–37. [CrossRef]
115. He, Y.; Liu, Y.; Wu, L.; Yang, Z.; Ren, K.; Qin, Z. MsDroid: Identifying Malicious Snippets for Android Malware Detection. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 2025–2039. [CrossRef]
116. Sawadog, Z.; Dembele, J.M.; Mendy, G.; Ouya, S. Zero-Vuln: Using deep learning and zero-shot learning techniques to detect zero-day Android malware. In Proceedings of the 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering, Tenerife, Canary Islands, Spain, 20–21 July 2023.
117. Liu, Y.; Tantithamthavorn, C.; Li, L.; Liu, Y. Deep Learning for Android Malware Defenses: A Systematic Literature Review. *ACM Comput. Surv.* **2023**, *55*, 1–36. [CrossRef]
118. Qiu, J.; Zhang, J.; Luo, W.; Pan, L.; Nepal, S.; Xiang, Y. A Survey of Android Malware Detection with Deep Neural Models. *ACM Comput. Surv.* **2020**, *53*, 1–36. [CrossRef]
119. Wang, S.; Wu, H.; Lu, N.; Shi, W.; Liu, Z. ATSDetector: An Android Trojan spyware detection approach with multi-features. *Comput. Secur.* **2025**, *150*, 104219. [CrossRef]
120. Le, T.H.M.; Babar, M.A. Automatic Data Labeling for Software Vulnerability Prediction Models: How Far Are We? In Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Barcelona, Spain, 24–25 October 2024.
121. Almomani, I.; Almashat, T.; El-Shafai, W. Maloid-DS: Labeled Dataset for Android Malware Forensics. *IEEE Access* **2024**, *12*, 73481–73546. [CrossRef]
122. Fei-Fei, L.; Fergus, R.; Perona, P. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 594–611. [CrossRef]
123. Li, D.; Lu, N.; Wang, S.; Shi, W.; Choi, C. A precise method of identifying Android application family. *Expert Syst.* **2024**, *41*, e13481. [CrossRef]
124. The European Parliament. Investigation of the Use of Pegasus and Equivalent Surveillance Spyware (Recommendation). Available online: https://www.europarl.europa.eu/doceo/document/TA-9-2023-0244_EN.html (accessed on 3 July 2025).
125. The White House of U.S.A. Joint Statement on Efforts to Counter the Proliferation and Misuse of Commercial Spyware. Available online: https://2021-2025.state.gov/joint-statement-on-efforts-to-counter-the-proliferation-and-misuse-of-commercial-spyware/ (accessed on 3 July 2025).