Scientific Research Publishing

International Journal of Intelligence Science, 2024, 14, 71-93
https://www.scirp.org/journal/ijis
ISSN Online: 2163-0356
ISSN Print: 2163-0283

Detection and Prevention of Malware in Android Mobile Devices: A Literature Review

Joseph Keteku[1], George Owusu Dameh[2], Samuel Ameka Mante[1], Thomas Kwashie Mensah[1], Schneider Laryea Amartey[1], John-Bosco Diekuu[3]

[1]ICT Directorate, University of Health and Allied Sciences, Ho, Ghana
[2]ICT Directorate, Simon Diedong Dombo University of Business and Integrated Development Studies, Wa, Ghana
[3]School of Computing, Robert Gordon University, Aberdeen, UK
Email: jketeku@uhas.edu.gh, godameh@ubids.edu.gh

How to cite this paper: Keteku, J., Owusu Dameh, G., Mante, S.A., Mensah, T.K., Amartey, S.L. and Diekuu, J.B. (2024) Detection and Prevention of Malware in Android Mobile Devices: A Literature Review. *International Journal of Intelligence Science*, **14**, 71-93.
https://doi.org/10.4236/ijis.2024.144005

Received: September 23, 2024
Accepted: October 27, 2024
Published: October 30, 2024

Abstract

Despite only being around for a few years, mobile devices have steadily risen to become the most extensively used computer devices. Given the number of people who rely on smartphones, which can install third-party apps, it has become an increasingly important issue for end-users and service providers to ensure that both the devices and the underlying network are secure. People will become more reliant on applications such as SMS, MMS, Internet Access, Online Transactions, and so on due to such features and capabilities. Thousands of devices ranging from low-cost phones to high-end luxury phones are powered by the Android operating system, which has dominated the smartphone marketplace. It is about making it possible for people from all socioeconomic backgrounds to get and use mobile devices in their daily activities. In response to this growing popularity, the number of new applications introduced to the Android market has skyrocketed. The recent appearance of a wide range of mobile malware has caught the attention of security professionals and scholars alike. In light of the ongoing expansion of the mobile phone industry, the likelihood of it being used in criminal activities will only continue to rise in the future. This article reviews the literature on malware detection and prevention in Android mobile devices, analyzes the existing literature on major studies and tasks, and covers articles, journals, and digital resources such as Internet security publications, scientific studies, and conferences.

Keywords

Android Malware, Android Mobile, Application Security, Malware Detection, Mobile Security

1. Introduction

Since the 1960s, telecommunications have been essential in influencing contemporary communication. Through ongoing study and technological progress, mobile computing has developed as a revolutionary technology, facilitating the wireless transmission of data, voice, and video. In the last 15 years, digital cellular technologies have significantly advanced, becoming essential instruments of the 21st century. These technologies have undergone swift expansion regarding the global user base, computational power, storage capacity, and functionality, even in more affordable variants. The advancement of resilient operating systems, compatibility with surrounding settings, and the accessibility of multifunctional applications have rendered mobile devices essential in numerous businesses and aspects of contemporary life.

The increased adoption of mobile devices has presented considerable security and privacy threats. Mobile devices, which manage substantial quantities of sensitive data, including Personally Identifiable Information (PII), have emerged as attractive targets for nefarious operations. This research seeks to enhance the current body of knowledge by evaluating the security and confidentiality concerns linked to digital applications and identifying probable illicit actions of certain organizations managing sensitive data.

Recent investigations have shown security vulnerabilities. For example, Uber has been documented to log users' geographical coordinates in a manner that breaches confidentiality agreements [1], yet governmental entities have successfully monitored individuals through applications such as Angry Birds [2]. TikTok, a highly popular application, was discovered to have been used by cybercriminals to collect smartphone IDs of their victims [3]. Moreover, third-party geographical searches and links to potentially harmful server domains have generated considerable security apprehensions [4] [5].

Researchers have also investigated additional security threats associated with mobile applications. [6] revealed the transmission of user data by advertising firms and smartphone app developers, whereas [7] examined the actions of harmful code and website surveillance. [8] and [5] investigated the implications of unprotected applications on geolocation inaccuracies and overarching security risks.

The growing dependence on mobile devices for daily management has introduced both ease and new difficulties. This study examines the literature on malware detection and prevention in Android mobile devices, utilizing information from security journals, scientific studies, and conferences to evaluate significant advancements and obstacles in this domain. This research highlights the necessity of enhancing mobile security in a progressively interconnected environment.

2. Theoretical Review

Telecommunications has been one of the most effective communications fields since the 1960s, with ongoing research leading to the emergence of mobile

computing as a technology that enables data, voice, and video transfer via wirelessly enabled devices. Over the previous 15 years, digital cellular technologies have evolved into one of the most critical technologies used in the 21st century. They have evolved from various angles while also experiencing a massive increase. These include the total number of users across the globe, the computing power available, the amount of storage space available, the development of increasingly complex functions in lower-priced models, the availability of robust operating systems, interoperability and recognition with surrounding environments, and the accessibility of applications that can be used to communicate with almost every facet of modernity in every industry. This research will add to the present body of knowledge by stressing the assessment of security and confidentiality risks connected to digital applications and highlighting the potentially illegal behaviours of some firms that handle sensitive Personally Identifiable Information.

New research, for example, has exposed how Uber records users' geographical coordinates in ways that violate business confidentiality agreements [1]. Another study has revealed how state authorities may utilise applications like Angry Birds to identify citizens [3]. TikTok, a highly popular application, was discovered to have been used by cybercriminals to collect smartphone IDs of their victims [9]. Additionally, it has been established that third-party geographical searches from applications may offer security risks [4] and many applications link to potentially malicious server domains [5]. Additionally, [10] exposed how advertising agencies and programmers of smartphone applications transport user data and [7] analysed dangerous code activities and website monitoring. Additionally, [5] examined clients' motivations for disclosing Personally Identifiable Information. According to [8] unsecured mobile applications can result in security concerns. [5] also addressed geolocation issues with mobile phone gadgets.

2.1. Security Model of Mobile Applications

Mobile application programming is at the heart of existing technological trends. These programs use a client-server paradigm to provide their services. A user using an iOS or Android-based smartphone acts as the client, while the application server is housed in a virtualised environment. In most circumstances, this server is primarily a web service with which the client's mobile application interacts via some Application Programming Interface (API). Thus, the cloud infrastructure, specifically the server, is a critical component of mobile applications because it is responsible for executing the program, connecting with the client for service provisioning, and interacting directly with the client device. **Figure 1** illustrates a typical context in which a mobile application connects to its server.

2.2. The Architecture of Mobile Applications

Well-known mobile application frameworks that have been used in the creation of applications. In most cases, this design can be broken down into three layers. A Data Layer is responsible for all data functions, access, modules, and service agents

handles the application's data. The Business layer oversees all business-related workflow and all business entities. Furthermore, the presentation layer allows the user to communicate with the application. This typical design is seen in Figure 2.

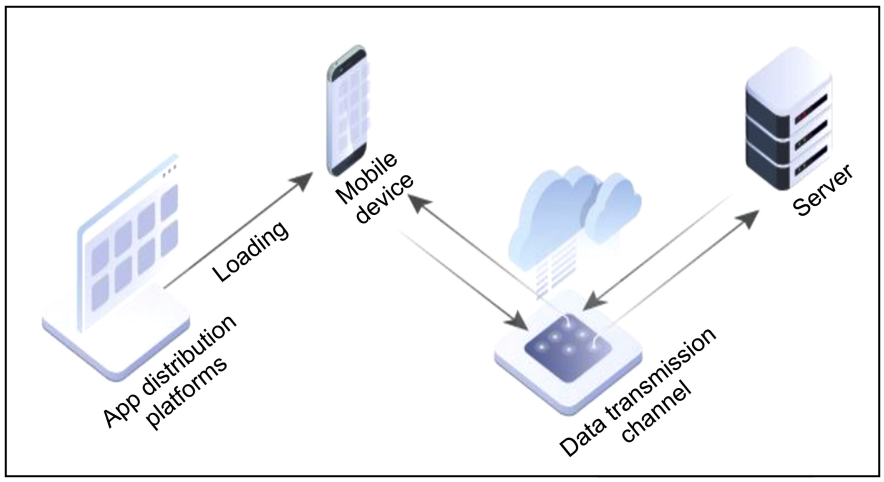

Figure 1. Mobile application infrastructure.

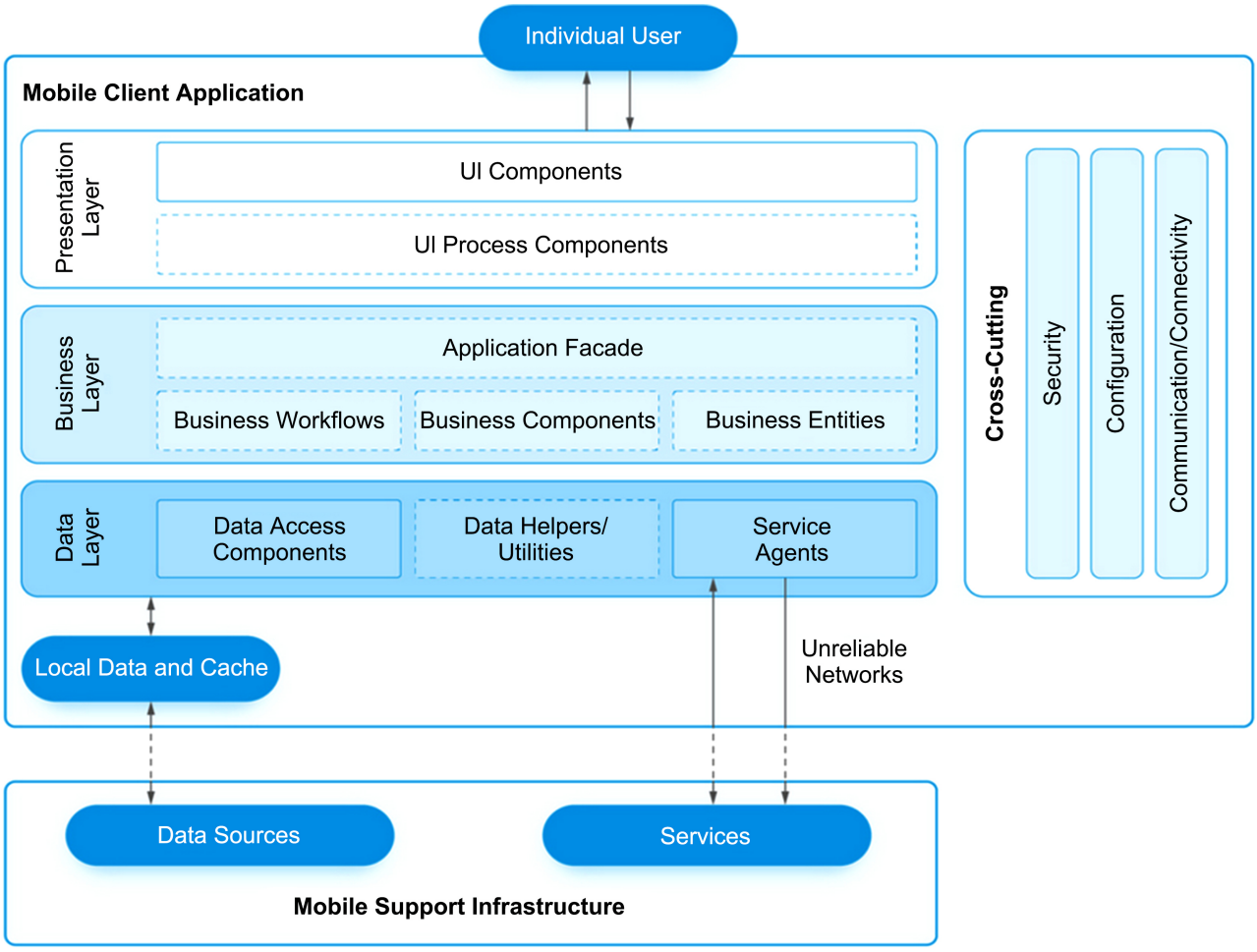

Figure 2. The architecture of mobile applications.

Several essential factors are considered when creating mobile application architecture, notably mobility, manageability, efficiency, and flexibility. Nevertheless, due to numerous occurrences involving the invasion of user privacy, security has emerged as a critical consideration when building and creating application architecture. It is regarded as the highest consideration before developing or building an application.

2.3. Application Architecture of Android Mobile

Linux is at the heart of the Android platform, which is also based on Linux like many other platforms. The Android operating system comprises five components, each serving a specific function. These elements function together so that the application layer is at the top, interacting with the user and providing a User Interface, aided by the application framework, and likewise with the assistance of the Android runtime and utilities located at the bottom. As depicted in **Figure 3**, the Linux kernel is the foundation for the Android system, allowing it to offer Android-based mobile applications.

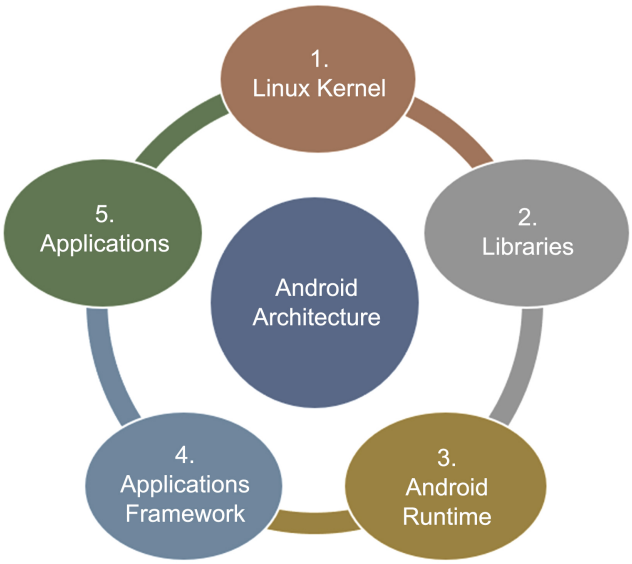

Figure 3. Application architecture of android mobile.

Security concerns led to using the Linux kernel in the Android operating system. The Linux operating system is an explicit operating system, making security straightforward to maintain because no hidden programs can run themselves without the user's permission.

2.4. Static Features of Android Applications

The manifest file is the glue that holds Android apps together when they are otherwise disjointed. The manifest file contains detailed information on each component, including any requirements for the system, any additional requirements, and any required permissions. Activity, service, content source, and broadcast

receiver are essential for an Android app. Individually, each of them serves a specific purpose. An APK file is a zip archive that contains assets, lib, res, manifest, Dalvik bytecode, and resource files for an Android program. When working with these files, it is common to turn to static feature sources. Each file's feature vector extraction method and expression are tailored to its intended use. **Figure 4** depicts the main traits that fall under each category [5].

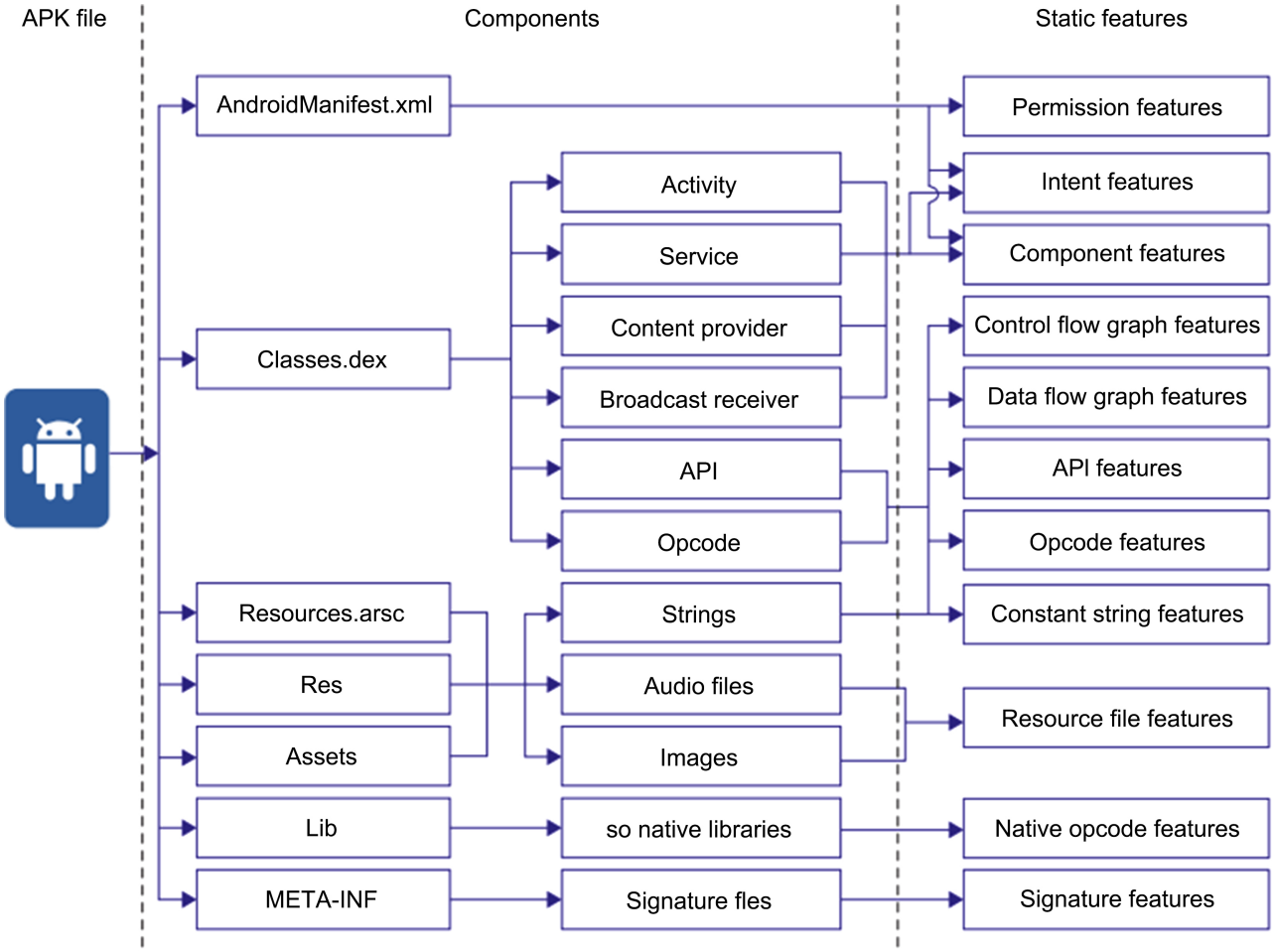

Figure 4. Android application's static features.

2.5. iOS Mobile Application Architecture

Regarding its UIKit, Apple relies on the MVC (Model View Controller) architecture for its IOS platform. Model is primarily concerned with data, i.e., its computation and storage through model objects, and so on. In this case, the view is responsible for the visual depiction of the application. Because it does not contain data, it can be repeated for numerous Application Programming Interfaces. While a controller sits between a model and a view and is essential for building a communication channel between the two, it interacts with abstraction through a framework. It is typically uninformed of the entities that are interacting with each other. **Figure 5** outlines the IOS mobile application Architecture.

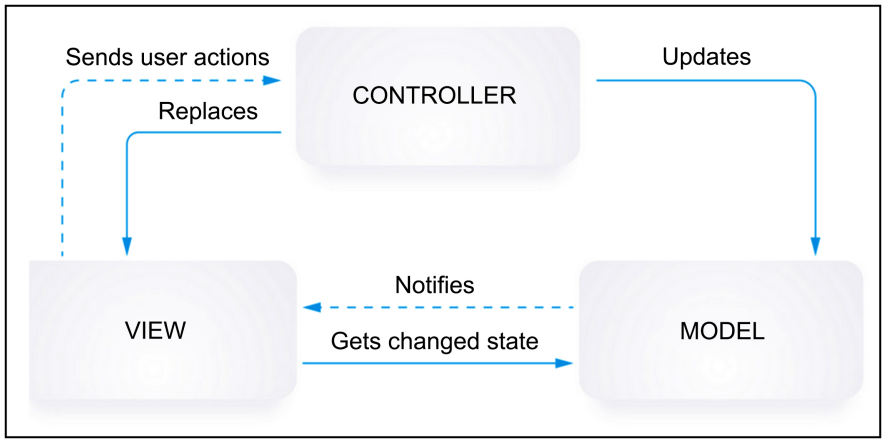

Figure 5. IOS mobile application architecture.

2.6. Security Flaws of Mobile Applications

Several surveys and research revealed hazardous flaws in Android and IOS mobile applications. Because of technological advancements, hackers no longer need physical access to the gadget to bypass security and steal sensitive information. Hackers can accomplish their illegal activities through security flaws present in mobile applications, which provide a backdoor entry to the smartphone, allowing the hacker to manipulate known security flaws and harm the user in various ways for evil reasons.

2.6.1. Client-Side Security Flaws

Most of the flaws are found on the client's side. It is estimated that around 60% of the flaws are in the client device. While 86% of the flaws may be leveraged without direct contact with the device, 16% require physical access. Android is more vulnerable than IOS regarding client-side security flaws, with Android receiving 43% of the vulnerability share while IOS only receives 38% of the vulnerability share. However, both operating systems have a high number of client-side vulnerabilities. One-third of those vulnerabilities have been determined to be extremely dangerous. Client-side security Flaws are presented in **Figure 6**.

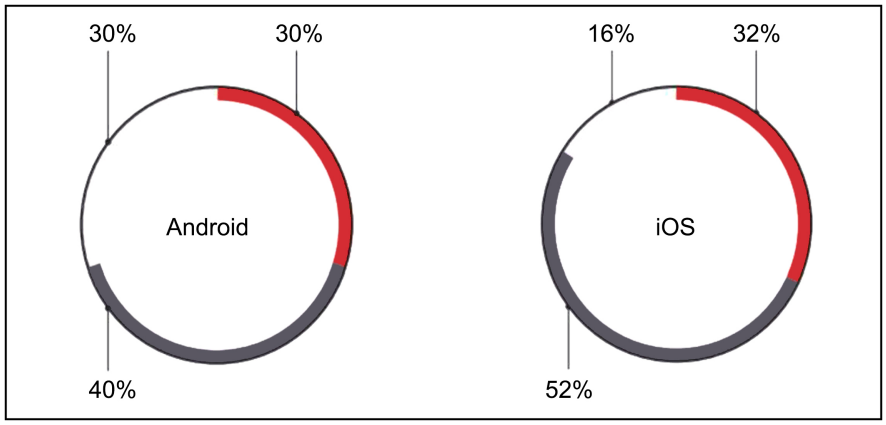

Figure 6. Client-side security flaws.

2.6.2. Server-Side Security Flaws

A web application that connects with the client-side through an application interface, commonly referred to as an API is used in most cases to provide services from the server. In the case of mobile applications, most of the vulnerabilities are found on the client side; however, due to the web-based nature of the server side, web applications are also highly vulnerable and have long been of interest to cybercriminals. 22% of the server-side flaws are considered extremely dangerous. Server-side security flaws are generally connected to the application security framework and application code. Server-side security flaws are presented in Figure 7.

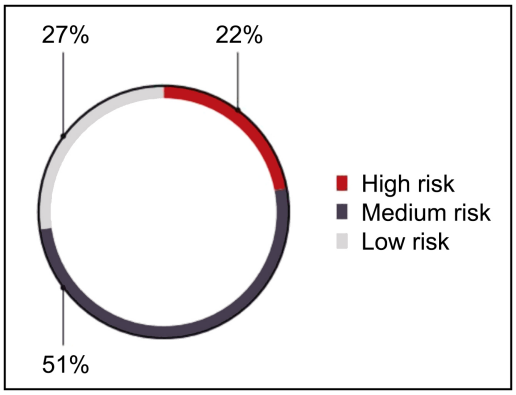

Figure 7. Server-side security flaws.

2.7. Examples of Android Application Malware

Malware, encompassing viruses, worms, trojans, and ransomware, is a detrimental program intended to infiltrate a device, network, or data. Android, an open-source platform, is especially susceptible to vulnerabilities owing to its open environment, vast user base, customization options, and protracted security patches. The open ecosystem of Android enables developers to distribute applications via third-party stores, hence heightening the danger of malware dissemination. The fragmentation of the platform, in contrast to iOS, permits delays in security updates, thereby exposing numerous devices to potential attackers. Consequently, Android serves as an optimal target for diverse forms of malware assaults.

Android devices are susceptible to numerous forms of malware, such as Trojan Horses, ransomware, spyware, adware, rootkits, keyloggers, and botnets. Trojan Horses masquerade as authentic software, whereas ransomware restricts user access or encrypts data. Spyware collects user data without approval, Keyloggers record keystrokes, capturing confidential information. Botnets link infected devices to hacked networks, frequently employed in coordinated assaults such as DDoS attacks.

Potential Threats of Malware on the Android Platform

Android malware presents possible risks including data theft, privacy violations, device takeover, money losses via phishing, deceptive applications, ransomware,

and diminished performance. These threats can expropriate sensitive personal information, provide remote control of compromised devices, incur substantial financial losses, and degrade device performance, leading to a diminished user experience and system instability.

2.8. OWASP Mobile Top Ten Android Application Flaws

The Open Web Application Security Project, or OWASP, is a non-profit organisation based in the United States established on April 21, 2004. The OWASP Foundation was established on December 1, 2001, and has since evolved into a global organisation that promotes all OWASP operations. The Open Web Application Security Project (OWASP) is a non-profit organisation dedicated to helping people better understand, design, procure, operate, and maintain secure apps. Any OWASP tool, publication, or forum can be used for free by anybody interested in improving and understanding application security. The OWASP Mobile Top 10 product covers the top ten vulnerabilities detected in mobile applications, according to OWASP. Table 1 shows the OWASP Mobile Top Ten, a list of the top ten most common faults detected in Android application development.

Table 1. OWASP mobile top ten android application flaws.

No	Vulnerabilities
1	Improper Platform Usage
2	Insecure Data Storage
3	Insecure Communication
4	Insecure Authentication
5	Insufficient Cryptography
6	Insecure Authorisation
7	Client Code Quality
8	Code Tampering
9	Reverse Engineering
10	Extraneous Functionality

2.8.1. Improper Platform Usage

Application programmers who fail to incorporate or poorly use a feature of the mobile operating system are said to be misusing platform [11]. This happens when a mobile app fails to follow "public rules," deviates from "standard procedure," or incorrectly implements a system feature [11]. "One-third of weaknesses in Android mobile applications arise from setup problems," according to a Positive Technologies analysis [12]. Touch ID and Keychain are two instances of security vulnerabilities caused by poor deployment [11]. Codified Security (2017) advocated "encrypting Keychain objects with 'This Device Only' values" to prevent

encrypted information from being revealed during backups and cloud syncing.

2.8.2. Insecure Data Storage

The second OWASP security issue that can harm smartphones is the data stored in an insecure manner (OWASP, 2018). According to a Positive Technologies fact sheet, 76% of mobile apps have this security threat [12]. The standard norm is that smartphone apps run in their own "sandbox" and can only read and edit their data. This misconception may have originated from this rule. Regrettably, that is not entirely accurate. Once a person gets physical access to a smartphone, exploiting vulnerable data is a simple process for that person to carry out.

First, a person who has discovered or hijacked a cellphone can connect the gadget straight to a computer and view the data instantly utilising free tools. This option is available to the individual regardless of whether the smartphone was found or stolen.

Secondly, the user can change a simple app or infect it with a virus to obtain the information on the cellphone. This kind of information can be found in repositories such as "SQL databases; Log scripts; XML data structures or manifest files; Binary data stores; Cookie stores; SD card; or Cloud synchronised" [13]. The risks associated with insecure data storage are directly proportional to the types of data processed and stored by an application. It is possible that storing data without any protection will not be a problem if the data in question is harmless and does not contain any personally identifiable information (PII) or private company data. On the other hand, if the opposite is true about sensitive data, then unsecured data storage can have significant negative repercussions, such as "identity theft; privacy violation; fraud; reputation damage; external policy violation; or material loss" [13].

2.8.3. Insecure Communication

Problems with transmission and communication between processes might be attributed to insecure communication. Positive Technologies (2019) reports that 35% of apps use insecure communication with external systems and 29% use insecure communication within their processes [12]. Using Transport layer security "to transfer private information," which includes session tokens; utilising CA trusted certificates and "denying self-signed certificates;" and "verifying the identity of the endpoint server using trusted certificates in the keychain;" are some of the controls that OWASP recommends for communication with external systems. Other controls include "using trustable certificates in the keychain to verify the identity of the endpoint server" [11]. It is feasible to protect oneself from attacks involving a man in the middle by first determining the interaction endpoint. In addition to securely delivering data, Interprocess communication is a security weakness with various attack paths. For example, Android apps can connect with other apps using "Intents," which provides a means for sending "broadcast" messages that any app can pick up [14].

If an app registers as a "Broadcast Receiver," it can obtain such messages, which

may include critical content [12]. In the same way, IOS 8 has an "Application Extensions" capability that enables "apps on the same device" to interact with one another [12]. According to Apple, app features like "Share," "File Provider," and "Document Provider" can be embedded within another app, an entry point if the data is not saved safely (Apple Developer, 2019). Positive Technologies (2019) mentions "deep linking" as another interprocess communication vulnerability where software might register itself to handle a specific URL. The issue is that any software can register itself to process the URL, which might lead to "link hijacking" and, eventually, a "phishing attack" [15]. [5] presented an example in which an application registers "fb" with a web address apart from Facebook and replicates a legitimate Facebook login to gather user credentials.

"App Link," was implemented in 2015 to combat link hijacking by offering "a means to authenticate the app-to-link association" [5]. When a user picks an "unauthenticated App link," the smartphone prompts the user to choose whether to use the internet browser or application, which can lead to link hijacking if the app is selected. Furthermore, [5] discovered another flaw in Android's implementation of "App Link," dubbed "an over permission vulnerability," that completely bypasses the prompting procedure.

2.8.4. Insecure Authentication

A smartphone application demonstrates insecure authentication, as defined by OWASP, when it interfaces with system services without submitting authentication, such as login details or an authentication token [16] Moreover, OWASP noted that authentication weaknesses are typically encountered when an app keeps user credentials directly on the device, when insecure accounts are permitted, or when Touch ID is employed [16].

Positive Technologies (2019) reported that 53% of applications do not save authentication data securely, and % use local authentication. [12] suggested that applications forgo client-side authentication but rather send encrypted login information to the server for validation to circumvent these vulnerabilities. Regarding applications that have weak passwords and limitless login attempts, brute-force assaults are problematic [9]. App lockout after several unsuccessful login attempts was suggested [9]. However, "two-factor authentication" is the best security against brute-force attacks, according to [17]. Because biometric comparisons are not flawless and can result in false positives, biometric authentication should include two-factor authentication [18].

2.8.5. Insufficient Cryptography

The Open Web Application Security Project (OWASP) stated that insufficient cryptography happens when an application makes use of a "flawed procedure" or a "poor algorithm" [16]. Jailbroken devices allow for the use of free tools that can decrypt encrypted applications [16]. When a smartphone gets jailbroken, it is no longer contained within a sandbox and instead has "unregulated accessibility to the full file system" [19]. A secure application becomes decrypted when it is put

into memory. At this point, a "snapshot" of the application can be captured and further studied using techniques associated with binary analysis, as stated by [11]. Because decryption is handled directly by the smartphone's operating system, this attack may be carried out with relative ease.

2.8.6. Insecure Authorisation

According to OWASP, insecure authorisation typically takes place when "Insecure Direct Object References (IDOR)" are present, "Concealed Data points" are employed, or unique privileges or rights are conveyed [11]. An IDOR can directly provide an object, such as a URL parameter, to execute an operation on the object [20]. [11] offered an example of submitting an "access token" and account ID to a backend service that does not correctly associate the token with the ID. This circumstance allows hackers to acquire illegal backend access permission to the wrong account by submitting a valid access token for another ID [11]. Hidden endpoints result from developers not bothering with authentication methods because they assume the feature is hidden from the user [11]. [12] also cites "session hijacking" as a further authorisation issue. Again, the issue pertains to client-server connections, in which the mobile app does not appropriately erase a session ID after it is closed [12]. Positive Technologies (2019) states that if the backend server does not impose session timeouts, an attacker can impersonate another user using an expired session ID. Positive Technologies (2019) reported that 35 % of apps have "inaccurate handling of session termination".

2.8.7. Client Code Quality, Code Tampering, Reverse Engineering and Extraneous Functionality

Poorly designed code, such as "buffer overflows," can also be a cause of security vulnerabilities in mobile applications, according to OWASP. This is the case with any application [16]. This weakness was separated from "incorrect platform utilisation" by [11] because it includes a programming error rather than an improper failure to implement an operating system capability. Regarding modifying source code, OWASP stated that applications, once installed on a mobile device, are susceptible to being tampered with [16].

Because of this, the Open Web Application Security Project (OWASP) recommended that applications test their compile-time rather than their runtime integrity. In a related vein, OWASP warned that applications are especially vulnerable to "reverse engineering due to the inherent nature of programming". OWASP suggested using "an obfuscation tool" that enables the programmer to specify "methods/code segments" and strings to obscure, to strike a balance between obfuscation and performance and to circumvent "de obfuscation" tools. All these measures are intended to prevent reverse engineering. Even though extraneous functionality is not unique to apps, OWASP included it in their list of mobile app vulnerabilities because it is common practice for developers to create "hidden backdoor functionality" for internal testing, which then inadvertently makes its way into the version of the app that is distributed to customers. Physical code

inspections are recommended as the most effective method for addressing this security issue by OWASP [16].

2.9. Related Work

2.9.1. An Investigative Framework Based on User Impulsivity and Secure Collaboration Errors

A scenario-based role-play study was conducted by [19] to determine how smartphone-registered users react to social engineering, primarily achieved through a mobile device. The researchers endeavour to use the study results for a controlled mobile phone forensics evaluation, which they assume will make it easier for digital forensics forensic experts to analyse data from smartphone devices. It was decided that a Quick Response code would be used for the study's purposes. The results established that legitimate Quick Response codes were handled better than fake ones, regardless of how well-informed the participants were about the experiment. Although educated people performed better with both sorts of Quick Response codes, they performed significantly better with the erroneous ones. Their research demonstrates that raising user awareness results in more dependable user behaviour on the user's part. Other factors that impact consumers when encountering a Quick Response code were also investigated.

Their inquiry demonstrated that controlled impulsiveness is associated with improved performance in managing incorrect Quick Response codes among users unaware of the study's purpose when they participated in the trial. The features of the participants had no effect when they were aware of the study's objective; on the other hand, their understanding of mobile device security aided them in distinguishing phoney Quick Response codes. Customers who are well-informed and aware of social engineering techniques are less prone to make hasty decisions about mobile device security, as demonstrated by this study. As a result, a forensic investigation technique based on smartphone users' most common security mistakes was developed and recommended. Because of the user's impatience, inadequate knowledge, and understanding, the forensics inquiry was advised to focus on the probable remains of user behaviours during the investigation. However, their research merely touched on permissions and malware on mobile devices and did not go into depth.

2.9.2. Discovering Reverse Engineering Signs in Android Apps

A straightforward way for detecting repackaging indications in android applications was revealed by [8] by exploiting a unique feature known as the String Offset Order (SOO), which Google developed. At the expense of current methodologies, SOO offers an accurate, fast, and dependable way to differentiate between original and repackaged code, even in obfuscation. Testing the method with commonly used obfuscation techniques demonstrates its durability and ability to detect repackaging irregularities in programs generated with apkTool and dalvikobfuscator, among other tools. AndroidSOO, in contrast to current methodologies, may evaluate applications in stages without the need for a training period, predetermined

detection patterns, or the inclusion of auxiliary libraries, among other things. Their findings also revealed that the proposed method is incredibly adaptive, as app analysis takes only a few milliseconds, demonstrating that it is highly versatile. They asserted that AndroidSOO is a viable tool that may be used as a complement to more thorough malware analysis.

2.9.3. Compile-Time Code Virtualisation for Android Applications

The responsiveness and effectiveness of reverse engineering are two critical challenges preventing reverse engineering from breaching copyrights on mobile applications and energy limits because of a decryption sequence. A novel technique for transferring code virtualisation from the DEX level to the native level has been proven by [21]. It is both safe and covert and very inexpensive. A pre-compilation step and a compile-time virtualisation step are both included in their approach. Pre-compilation is meant to improve efficiency by identifying and decompiling essential routines that take up a significant amount of the execution time during the program. Compile-time virtualisation is built on the popular LLVM compiler technology used in many other applications. It translates DEX bytecode into the normal LLVM intermediate representations in seconds, allowing a unified code virtualisation to pass for DEX code to be applied immediately after. They developed a working Dex2VM prototype of their technique and put it through its paces on eight different Android applications.

Their testing results indicated that the recommended technique might effectively hide the target code against a cutting-edge code reverse engineering tool explicitly created for code virtualisation. It can do so at a minimal cost while maintaining high stealth. The Dex2VM prototype is presented in **Figure 8**.

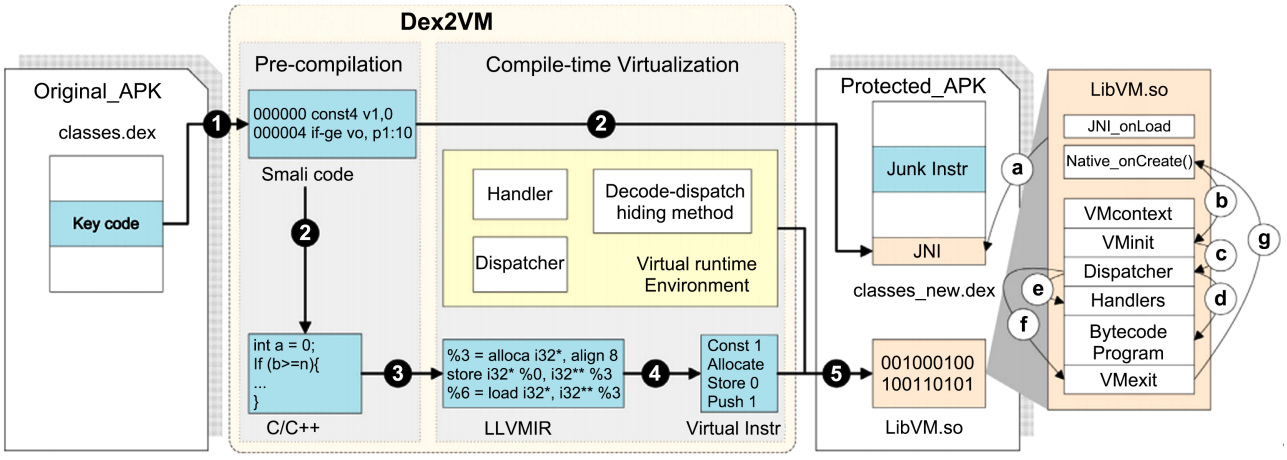

Figure 8. Functional Dex2VM prototype.

2.9.4. A TAN Based Hybrid Model for Android Malware Detection

Based on the analysis of environmental interdependence, [20] developed a novel technique for identifying Android malware applications that mix static and dynamic factors that influence dangerous behaviour by assessing static and dynamic

characteristics. The authors presented a novel hybrid malware detection technique based on TAN (Tree Augmented Naive Bayes) that capitalises on provisional correlations among vital static and dynamic characteristics (API calls, permissions, and system calls) required for an application's operation. To determine whether an application is risky, the researchers conditioned three ridge regularised logistic regression classifiers corresponding to the program's API calls, permissions, and system calls and modelled their output associations as a TAN (Tree Augmented naive Bayes). The tests revealed that the proposed method could detect malicious applications with an accuracy of 0.97 over an extended period. The proposed technique catches dangerous behaviour more accurately than conventional static and dynamic analysis tools, which are currently available.

However, a small number of malware programs are capable of evading detection by employing adversarial strategies. Because of this, a future direction for the research is to develop more powerful Bayesian models for efficiently spotting malicious malware programs through reinforcement deep learning techniques.

2.9.5. Anomaly Detection of Android Malware Using One-Class KNearest Neighbours (OC-KNN)

[22] did substantial research into One-Class Classification algorithms for Android malware anomaly detection as a countermeasure to the growing number of unknown malware varieties. In the case of novel viruses, the One-Class k-Nearest Neighbours techniques were proven effective when applied and generated using unary attributes from especially attacking patterns. The results were essentially good, with an accuracy rate of 88.5% for outlier detection, a rate of 2.4% for errors, and a rate of 1.14% for false alarms, all of which are exceptional. The results indicate that the false-positive rate was shallow and negligible, indicating that the normalcy model had excellent detection accuracy. Thus, the research concludes with the proposal of a unique approach based on One-Class Classification for detecting unknown Android malware. While they contributed to the Android malware detection knowledge base, they did not focus on expanding feature sets to include APIs for benign applications to provide a more diverse feature sampling for Android malware detection.

2.9.6. Static and Dynamic Analysis of Android Malware

Machine learning approaches were utilised to discover which attributes were the most effective while examining the comparative effectiveness of several static and dynamic properties for recognising Android malware by [23]. The efficiency of the scoring systems under consideration is also considered. According to the researchers, scholars discovered that when it comes to Malware identification, an essential static feature based on permissions is far more informative than a dynamic feature that relies on system calls. System calls are not regarded as the gold standard for malware detection, which is surprising. System calls are believed to be the gold standard for malware detection in most malware detection literature, including this one. According to [23] even a slight reduction in the number of

privileges can result in considerable benefit from the perspective of the malware developer. Monkey Runner was used to gather their dynamic characteristics, which may have failed to execute the damaging bits of the code since it could not run them properly.

Using a more advanced way of extracting system calls may result in improved detection findings but at the expense of higher complexity and work.

2.9.7. Scandal

SCANDAL is a Static Analyser developed by [24] to detect privacy leaks in Android applications. The methodology known as the static analyser SCANDAL is a method that delivers a formally sound and automatic static analysis. It has been described as a reliable and automated static analyser to locate personal information leaks in Android applications. This program assessed 90 popular applications using SCANDAL from the Android Market and discovered privacy leaks in 11 applications. Additionally, this tool analysed eight known harmful applications from third-party markets and found privacy leaks in all eight applications. One of these model's drawbacks is that SCANDAL does not provide complete support for reflection-related APIs. Additionally, the time efficiency and memory utilisation during the analysis are poor, which should be considered for future efforts.

2.9.8. A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices

[17] offered ways to detect Android viruses based on static analysis. These approaches include permission-based detection, signature-based detection, and Dalvik bytecode detection. They also analysed the advantages and disadvantages of their respective methods. In addition, there was no comprehensive report, and the total number of studies included was 27, but there was neither a discussion nor a presentation of the work that would be done in the future.

2.9.9. DroidData

DroidData, which tracks and monitors data transfer within the Android operating system, was developed by [3]. DroidData is an innovative new tool that tracks and monitors data transfer within Android applications by employing both static and dynamic analysis. This strategy reduces the number of false positives while simultaneously increasing the amount of code coverage to find the most significant number of data breaches. Because of this new way of combining two different sorts of studies, DroidData can catch more data transmissions than other works that have been done previously. A user interface that is both adequately built and simple to use allows consumers to comprehend data transmissions and stop programs that unacceptably send their details. Results regarding communications are more comprehensive than those produced by other tools, offering information such as the level of safety associated with data transmission. On the other hand, DroidData, along with the vast majority of the different analysis tools currently available for Android, cannot track implicit data flows or regulate data flows.

Because malicious code often uses implicit flows to circumvent detection and take advantage of security features, this will need to be guarded against in the future.

2.9.10. PiOS

[25] developed PiOS for Detecting Privacy Leaks in iOS Applications. PiOS uses static analysis to check apps to determine whether they have code pathways in which an application initially accesses sensitive data and then eventually sends this data across a network. Because there is no source code available, PiOS is forced to do its analysis solely on the bytecode. PiOS determines these registers' values by using backward slicing at every call site to the objmsgSend function in a program package. PiOS identifies programs written for iOS that leak private information. PiOS can ascertain the kind of receiver used (R0) and the value that is being used for the selector (R1). It annotates the call site with the specific class and method invoked when the program is run, and it does this by putting the information in brackets. Obtaining the decryption key for the application from the device is not a simple operation (or the secure key chain of the operating system).

In addition, to use these keys, one would need first to develop the appropriate decryption methods in their system. During execution, PiOS does not record (the addresses of) the individual instances of allocated classes. Additionally, it is not always possible to statically determine the receiver and selection for every call made by the objcmsgSend method.

2.10. Advantages and Disadvantages of Existing Malware Detection and Prevention Technologies

2.10.1. Static Analysis

Static analysis is quick and able to find known dangerous patterns since it entails looking at the code of an application without running it. Applying it before the program is launched helps stop malware from getting into the system since it requires less resources.

Static analysis suffers with obfuscated code and polymorphic malware changing their signatures to hide detection. It is also less successful in spotting zero-day or unidentified threats, which may be needed for more dynamic, behavior-based investigation.

2.10.2. Dynamic Analysis

Dynamic analysis looks at an app's behavior during operation, therefore enabling it to identify potentially harmful activity not apparent in the code. It is quite good in identifying runtime events such odd network activity, privilege escalation, or data leaking.

Because dynamic analysis runs the software in a sandbox or virtual environment, it is more resource intensive. It can also be time-consuming and might overlook the malware that operates normally until specific criteria are satisfied (e.g., on a particular device or following a delay).

2.10.3. Machine Learning-Based Detection

By learning patterns and behaviors from big datasets, machine learning (ML) approaches can identify known as well as unknown malware. As more data is exposed to these models, they can evolve over time and provide great accuracy in identifying malware, including novel variations.

One of the drawbacks of ML models is great reliance on the quality and volume of training data. They can also be susceptible to adversarial attacks, in which case malware is developed especially to evade ML-based detection. Furthermore, challenging to apply real-time scenarios on mobile devices with limited capacity, these models need large processing resources.

2.10.4. Hybrid Approaches

Often using the qualities of both methods, hybrid strategies mix static and dynamic analysis and often produce greater detection rates. By examining both the code and behavior of apps, they can offer thorough coverage that raises the possibility of identifying advanced threats.

Considered typically more complicated and resource-intensive, these techniques call for both the efficiency of fixed methods and the computing expense of dynamic analysis. Because hybrid systems must combine two different approaches, their deployment might be difficult.

3. Methodology

Several crucial procedures were taken during the selection phase to guarantee that the literature assessment was thorough and methodical.

Initially, a comprehensive search was carried out using major academic databases, such as Scopus, Web of Science, ScienceDirect, IEEE Xplore and Google Scholar. These databases were picked because they provide a wide range of pertinent studies on malware detection, mobile security, and cybersecurity. Specific keywords, such as "Android malware detection," "mobile malware prevention," "Android security threats," "malware analysis on mobile devices," and "malicious software detection in Android," were used to narrow the search and locate the most pertinent research. Boolean operators such as "AND" and "OR" were also used to combine terms and get more precise results. Only publications released between 2010, and the present were selected to maintain the review's focus on the most current advancements in the field.

Several criteria were used to aid the publication's selection process. Initially, publications were selected according to how well they addressed Android device malware detection and prevention. To keep the research objectives sharply focused, studies that addressed Android malware but otherwise concentrated on general mobile security were disregarded. To make sure that the review represented the most recent developments and trends in Android malware detection and prevention, recent publications. Especially those published in the last ten years (2014 to present) were given priority. Additionally, the information supplied was credible and reliable since only high-quality sources such as conference

papers, peer-reviewed journals, and technical reports were taken into consideration.

Particular attention was paid to empirical research that presented data and findings from real-world settings about the efficacy of different malware detection and prevention strategies. These studies provide useful understanding of the approaches' effectiveness. The review additionally aimed to encompass a wide variety of methods for detecting malware, including machine learning-based approaches, hybrid methods that combine both static and dynamic analyses, and static analysis, which examines code without running it.

The study guarantees a methodical and thorough assessment of the most pertinent literature by adhering to these search tactics and selection criteria, collecting the most recent developments and the most successful methods for identifying and preventing malware on Android devices.

4. Discussion

Mobile malware investigations have shown to be challenging to conduct successfully using conventional forensics tools and procedures [26]. Users are increasingly storing their private information on mobile devices, which they do through various popular applications such as e-commerce, finance, and social media platforms. Intruders have shifted their focus to mobile apps, particularly the last decade [14]. As a result, Android malware has unquestionably risen to become one of the most severe security risks in information security. [27] iterated that the question of how to detect Android malware has become a serious one. Customers have come to demand a safe and secure environment, which is provided by the application stores. In other words, they trust their app sources are trustworthy and secure enough. Small businesses and personal computers are incredibly vulnerable to malware attacks because of the poor quality of their online applications.

The unethical exploitation of user information or the imposition of additional costs is a financially viable but unethical growth strategy [9]. Android malware has also been developed because of its large market share. There are more and more examples of malicious software in the wild every year [28]. Antivirus companies, market retailers, and even computer programmers have all worked to improve malware detection through cryptography [29]. Malware's evolution has been greatly aided by the emergence of cryptographic technologies in software applications [26]. Most malware is installed unintentionally by users [26]. Programs downloaded from third-party application shops, and numerous others. These unofficial stores sell their repackaged applications. A modified version of the original applications that are available on Google PlayStore. Cabir, the first mobile malware, was discovered in the Symbian mobile operating system in 2004. Since then, there has been a steady growth in the number of malwares in the mobile market [4].

Numerous strategies are employed to identify malware on mobile architecture. According to the Federal Bureau of Investigation (FBI), mobile device crimes

currently account for more than 70% of all crimes committed [15]. Even though cell phones are implicated in many crimes, investigations into information crimes have grown increasingly important. The Open Web Application Security Project (OWASP) publishes a top ten list of web application and mobile security vulnerabilities. It reviews them every year, including an in-depth study and specific information regarding the most widespread hazards among smartphone applications. The report covers a variety of subjects, including communication and authorisation, as well as storage, authentication, and encryption. Developers should use the report to address the vulnerabilities as applicable to the mobile environment from a client-server point of view. The OWASP literature offers information to developers regarding core security vulnerabilities associated with smartphone apps and provides suggestions on how to prevent such dangers.

5. Directions for Future Research and Development

Future research ought to focus on advancing lightweight, real-time malware detection systems capable of functioning effectively on mobile devices that possess constrained resources. This encompasses the refinement of machine learning models to ensure they operate efficiently on smartphones, minimizing battery consumption and processing demands.

Given the ongoing evolution of malware development strategies aimed at evading detection, it is imperative that upcoming research prioritizes the enhancement of detection methodologies to effectively address challenges such as code obfuscation, polymorphism, and adversarial attacks. Recent developments in deep learning and neural networks present intriguing possibilities for constructing more resilient systems that can comprehend and adjust to the changing tactics of malware.

There is a growing fascination with detection methods that prioritize the observation of real-time application behavior rather than merely examining the underlying code. Investigating this domain has the potential to enhance malware detection, rendering it more agile and proficient in recognizing zero-day vulnerabilities and novel threat types that frequently evade conventional static analysis techniques.

The proliferation of mobile devices handling extensive personal data necessitates the creation of detection systems that uphold user privacy principles. Future research may investigate the optimal equilibrium between efficient malware detection and the reduction of data collection, thereby safeguarding privacy while upholding security standards.

Cloud computing presents intriguing opportunities for the detection of malware. Delegating intricate analytical tasks to the cloud has the potential to alleviate the strain on mobile devices, thereby enhancing the speed and efficacy of detection systems. Furthermore, employing collaborative methodologies, in which malware information is disseminated across various devices and networks, has the potential to bolster early detection and refine response times.

The potential of blockchain technology to revolutionize the prevention of Android malware lies in its capacity to facilitate decentralized security solutions. Future research may investigate the potential of blockchain technology to authenticate the integrity of applications and establish tamper-proof records of identified malware, thereby fostering a more secure ecosystem.

Confronting these challenges and delving into these research avenues will pave the way for the development of more sophisticated, efficient, and secure malware detection and prevention systems for Android devices in the future.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Rastogi, S., Bhushan, K. and Gupta, B.B. (2016) Android Applications Repackaging Detection Techniques for Smartphone Devices. *Procedia Computer Science*, **78**, 26-32. https://doi.org/10.1016/j.procs.2016.02.006

[2] Xu, C. (n.d.) Android Ransomware Trends and Case Studies: A Reverse Engineering Approach. 55.

[3] Pandela, T. and Riadi, I. (2020) Browser Forensics on Web-Based Tiktok Applications. *International Journal of Computer Applications*, **175**, 47-52. https://doi.org/10.5120/ijca2020920897

[4] Goni, I., Gumpy, J.M., Maigari, T.U. and Mohammad, M. (2020) Cybersecurity and Cyber Forensics: Machine Learning Approach Systematic Review. *Semiconductor Science and Information Devices*, **2**, 25-29. https://doi.org/10.30564/ssid.v2i2.2495

[5] Wu, Q., Zhu, X. and Liu, B. (2021) A Survey of Android Malware Static Detection Technology Based on Machine Learning. *Mobile Information Systems*, **2021**, Article ID: 8896013. https://doi.org/10.1155/2021/8896013

[6] Amro, B. (2017) Malware Detection Techniques for Mobile Devices. *International Journal of Mobile Network Communications & Telematics*, **7**, 1-10. https://doi.org/10.5121/ijmnct.2017.7601

[7] Kaushik, P. and Jain, A. (2015) Malware Detection Techniques in Android. *International Journal of Computer Applications*, **122**, 22-26. https://doi.org/10.5120/21794-5166

[8] Gonzalez, H., Kadir, A.A., Stakhanova, N., Alzahrani, A.J. and Ghorbani, A.A. (2015) Exploring Reverse Engineering Symptoms in Android Apps. *Proceedings of the* 8*th European Workshop on System Security*, Bordeaux, 21 April 2015, 1-7. https://doi.org/10.1145/2751323.2751330

[9] Gope, P. (2019) LAAP: Lightweight Anonymous Authentication Protocol for D2D-Aided Fog Computing Paradigm. *Computers & Security*, **86**, 223-237. https://doi.org/10.1016/j.cose.2019.06.003

[10] Amro, B. and Abu Znaid, Z. (2021) User Centric Android Application Permission Manager. *Al-Rafidain Journal of Computer Sciences and Mathematics*, **15**, 213-223. https://doi.org/10.33899/csmj.2021.170043

[11] OWASP Top Ten 2017|2017 Top 10|OWASP Foundation (n.d.). https://owasp.org/www-project-top-ten/2017/Top_10

[12] Cybersecurity Threatscape 2019 (n.d.).

https://global.ptsecurity.com/analytics/cybersecurity-threatscape-2019

[13] Introduction—OWASP Top 10 Proactive Controls (n.d.). https://top10proactive.owasp.org/archive/2018/0x04-introduction/#:~:text=The%20OWASP%20Top%20Ten%20Proactive,those%20new%20to%20secure%20development

[14] Alkindi, Z.R., Sarrab, M. and Alzeidi, N. (2021) User Privacy and Data Flow Control for Android Apps: Systematic Literature Review. *Journal of Cyber Security and Mobility*, **10**, 261-304. https://doi.org/10.13052/jcsm2245-1439.1019

[15] Darabian, H., Homayounoot, S., Dehghantanha, A., Hashemi, S., Karimipour, H., Parizi, R.M., *et al.* (2020) Detecting Cryptomining Malware: A Deep Learning Approach for Static and Dynamic Analysis. *Journal of Grid Computing*, **18**, 293-303. https://doi.org/10.1007/s10723-020-09510-6

[16] Alanda, A., Satria, D., Mooduto, H.A. and Kurniawan, B. (2020) Mobile Application Security Penetration Testing Based on OWASP. *IOP Conference Series*: *Materials Science and Engineering*, **846**, Article ID: 012036. https://doi.org/10.1088/1757-899x/846/1/012036

[17] Feng, R., Chen, S., Xie, X., Meng, G., Lin, S. and Liu, Y. (2021) A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices. *IEEE Transactions on Information Forensics and Security*, **16**, 1563-1578. https://doi.org/10.1109/tifs.2020.3025436

[18] Narwal, B. and Goel, N. (2020) A Walkthrough of Digital Forensics and Its Tools.

[19] Babun, L., Sikder, A.K., Acar, A. and Uluagac, A.S. (2019) A Digital Forensics Framework for Smart Settings. *Proceedings of the* 12*th Conference on Security and Privacy in Wireless and Mobile Networks*, Miami, 15-17 May 2019, 332-333. https://doi.org/10.1145/3317549.3326317

[20] Surendran, R., Thomas, T. and Emmanuel, S. (2020) A TAN Based Hybrid Model for Android Malware Detection. *Journal of Information Security and Applications*, **54**, Article ID: 102483. https://doi.org/10.1016/j.jisa.2020.102483

[21] Zhao, Y., Tang, Z., Ye, G., Peng, D., Fang, D., Chen, X., *et al.* (2020) Compile-Time Code Virtualization for Android Applications. *Computers & Security*, **94**, Article ID: 101821. https://doi.org/10.1016/j.cose.2020.101821

[22] Gyunka, B.A. and Barda, S.I. (2020) Anomaly Detection of Android Malware Using One-Class K-Nearest Neighbours (OC-KNN). *Nigerian Journal of Technology*, **39**, 542-552. https://doi.org/10.4314/njt.v39i2.25

[23] Kapratwar, A., Di Troia, F. and Stamp, M. (2017) Static and Dynamic Analysis of Android Malware. *Proceedings of the* 3*rd International Conference on Information Systems Security and Privacy*, Vol. 1, 653-662. https://doi.org/10.5220/0006256706530662

[24] Ahsen, M., Hassan, S.A. and Jayakody, D.N.K. (2016) Propagation Modeling in Large-Scale Cooperative Multi-Hop Ad Hoc Networks. *IEEE Access*, **4**, 8925-8937. https://doi.org/10.1109/access.2016.2635718

[25] Egele, M., Kruegel, C., Kirda, E. and Vigna, G. (n.d.) PiOS: Detecting Privacy Leaks in iOS Applications.

[26] Alashjaee, A.M. and Haney, M. (2021) Forensic Requirements Specification for Mobile Device Malware Forensic Models. 2021 *IEEE* 11*th Annual Computing and Communication Workshop and Conference* (*CCWC*), 27-30 January 2021, 930-935. https://doi.org/10.1109/ccwc51732.2021.9376043

[27] Sharma, B.K., Joseph, M.A., Jacob, B. and Miranda, B. (2019) Emerging Trends in

Digital Forensic and Cyber Security—An Overview. 2019 6*th HCT Information Technology Trends* (*ITT*), Ras Al Khaimah, 20-21 November 2019, 309-313. https://doi.org/10.1109/itt48889.2019.9075101

[28] Alzaylaee, M.K., Yerima, S.Y. and Sezer, S. (2020) Dl-Droid: Deep Learning Based Android Malware Detection Using Real Devices. *Computers & Security*, **89**, Article ID: 101663. https://doi.org/10.1016/j.cose.2019.101663

[29] You, G., Kim, G., Cho, S. and Han, H. (2021) A Comparative Study on Optimization, Obfuscation, and Deobfuscation Tools in Android. *Journal of Internet Services and Information Security*, **11**, 2-15. https://doi.org/10.22667/JISIS.2021.02.28.002