ASIAN BULLETIN OF BIG DATA MANAGMENT
Vol. 4. Issue 4 (2024)
https://doi.org/ 10.62019/abbdm.v4i4.252

ASIAN BULLETIN OF BIG DATA MANAGEMENT
http://abbdm.com/
ISSN (Print): 2959-0795
ISSN (online): 2959-0809

Android Security Vulnerabilities, Malware, Anti-Malware Solutions, and Evasion Techniques

Mahtab Khalid, Ahthasham Sajid*, Muhammad Usman, Mehak Saeed, Malik Muhammad Nadeem, Ishu Sharma

Chronicle

Article history
Received: 21st November, 2024
Received in the revised format: 1st December, 2024
Accepted: 18th December, 2024
Available online: 21st December, 2024

Mahtab Khalid, Ahthasham Sajid, Muhammad Usman, Mehak Saeed and Malik Muhammad Nadeem are currently affiliated with Department of Information Security and Data Science, Riphah Institute of Systems Engineering, Riphah International University Islamabad, Pakistan.
Email: mqazimahtab1162@gmail.com
Email: ahthasham.sajid@riphah.edu.pk
Email: usman.s@oulook.com
Email: mehaksaeed1009@gmail.com
Email:nadeemsandila1989@gmail.com

Ishu Sharma is currently affiliated with Department of Computer Science and Engineering, Chandigarh Engineering College, Chandigarh Group of Colleges Jhanjeri, Sahibzada Ajit Singh Nagar, Punjab, India
Email: ishu.sharma001@gmail.com

Corresponding Author*

Abstract

This paper investigates the vulnerabilities inherent in the Android operating system architecture and examines how malware developers exploit these weaknesses to execute a variety of attacks. These include aggressive advertising, remote control capabilities, financial fraud, privilege escalation, and the leaking of sensitive information. In this paper, we survey a collection of anti-malware techniques and organize these techniques into three canonical classes (static methods, dynamic methods, hybrid methodologies) according to how they are used or implemented with respect to the host operating system. We also evaluate the effectiveness of these techniques against certain types of attacks and summarize them under test categories for reporting results. We also examine the typical countermeasures used by malware authors to disguise their approaches against existing detection methods, such as reintegrating with real applications, using update payloads, executing dynamic code, scrambling dangerous content, and setting traps to act upon only purposely triggered situations. In future work, we suggest research into the capability of reinforcement learning methods to further increase sustainability and adaptability of anti-malware strategies. This Research study aimed at creating more dynamic detection systems for malware by utilizing machine learning techniques that could change in parallel with the tactics used by malware developers. By leveraging this technique, you could drastically boost the efficacy of existing anti-malware solutions that are unable to respond to new threats. As the digital environment changes (and continues to change), Mobile provides threat analysts, fraud / security managers and legal authorities up-to-date circumstantial direction so users can move with confidence within their mobile landscapes.

Keywords: Android Malware detection, Permission on Android, Dangerous Permission, Malicious Code on Android.

© 2024 EuoAsian Academy of Global Learning and Education Ltd. All rights reserved

INTRODUCTION

Mobile phones, first introduced into our daily lives during the 1990s, were initially created for the purposes of sending text messages and making phone calls. However, with advancements in technology and the rise of mobile internet, they have evolved to enable a wide range of tasks with ease. Smartphones have become indispensable tools for daily tasks, from shopping and reading news to handling banking transactions and staying connected via social media, making them a crucial part of contemporary life. Smartphones run on a variety of operating systems such as Android, iOS, Samsung, KaiOS, BlackBerry OS, Tizen, and Windows Mobile. By the close of 2022, Android dominated the market, with a share of 71.75%. Projections for 2023 estimate around 3.6 billion active Android users worldwide across 190 countries. Android holds 70.94% of the global mobile operating system market, while Apple's iOS accounts for 28.33% Turner, A. (2022). Google developed Android, a rapidly growing, open-source, and fully customizable mobile operating system. It offers an open platform that allows Original Equipment Manufacturers (OEMs) like Samsung, Xiaomi, Oppo, Vivo, Huawei, Motorola, and Google full access and control over the system. This flexibility has allowed these manufacturers to offer devices at significantly lower price points, especially when compared to the average sale price of Apple iOS devices, which was $261 in the fiscal year 2021. This pricing strategy is a key factor in Android's widespread success. Furthermore, the Android operating system is widely used in smartphones, wearable devices, and smart TVs Android TV. (n.d.). With the growing number of app downloads from platforms like Google Play and the App Store, concerns about security have become more prevalent. Malware, which has long affected computers, is now infiltrating smartphones. Malicious software can result in various damaging consequences, such as unauthorized access to personal data, monitoring of user activities and locations, hacking of social media accounts, breaches of banking information, unauthorized message sending, and a decrease in both memory capacity and battery life Marko M. (2019, November 15). The rapid expansion of Android applications, coupled with its position as the leading operating system, has made it a major target for malicious software.

As technology continues to advance and reshape our daily lives, the number of smartphone and smart device users for both personal and business purposes is rapidly growing. Android, the most widely used operating system in this domain, accounts for 87% of users Nick Jasuja. (2019). Many smart device manufacturers back Android, which was first launched in September 2008 as a Linux-based, open-source platform supporting over 100 languages. Android apps are widely available through multiple app stores, such as Google Play, Amazon, Aptoide, and Galaxy Store. The vast selection of millions of apps has played a significant role in Android's popularity over other operating systems (2024). Av-Test.org. Yet recently, as the amount of apps and users has grown considerably, Android APKs have increasingly become a prime destination for greedy hackers looking to profit. Other — According to the latest AVTest security report an average of over 600,000 malware applications are being distributed each month. The report identifies common vulnerabilities and reveals trends based on an analysis of attack patterns.

As one example, the 2019 report highlighted that the majority of attacks used hardware architecture vulnerabilities to access memory content and could compromise sensitive data such as passwords. It also puts a year-by-year lens on the kind of attacks that have been launched, with 2019 being a particularly bad year for Windows devices specifically targeting them. A more substantial proportion of the sort of attacks that we saw targeted at Android OS reported each year, in both numbers and types, a wide range of threat levels. According to AVTest. This emerging threat highlights the importance of a systematic study of malware behaviors for creating high-performance detection and classification methods Kivva, A. (2023, June 7). So, in the following section, we will now concentrate on Android malware- its different forms and what strategies we could follow for defending against that malicious software.

In many cases it is impossible to carry out adequate security checks on all the applications released due to the high rate at which they are developed, so some apps will make it through time and again with unnoticed embedded vulnerabilities that can compromise the devices of end-users. Malware developers are also becoming more creative when it comes to hiding malicious payloads inside complex GUI widgets small app views that can be embedded in other apps (think of one your home screen weather widget updates, but rounds up to a very big number). This makes it difficult for malware analysis tools to identify and analyze the hidden threats. Please realize that the processing power, storage and limited battery life of the smart devices are very small and these traditional "PC anti-malware" techniques require a huge amount of resources. Moreover, traditional anti-malware methods, mainly based on signature-based antivirus (AV) solutions, are inadequate for detecting and preventing new malware variations since malware continually modifies its signature patterns. As these threats evolve, they necessitate more sophisticated detection strategies. Signature-based AV systems depend on a fixed set of malware characteristics for identification and classification, but malware can easily evade these defenses using techniques such as obfuscation or encryption. Consequently, alternative anti-malware techniques, including static and dynamic analysis, are being increasingly employed to combat these advanced attacks.

As reported by the Kaspersky Security Network, approximately 4,948,522 attacks involving mobile malware, adware, and risky software were blocked in the first quarter of 2023. Adware constitutes the most common threat to mobile devices, accounting for 34.8% of all detected threats Mathur, A et.al (2021). . In recent years, the Android operating system (AOS) has rolled out multiple updates to tackle various security vulnerabilities Mathur, A. (2022). The main defense mechanism in Android is Google Play Protect, which detects and mitigates malicious applications found in the Google Play Store. However, many third-party app stores allow users to download potentially harmful software. Furthermore, the Android OS utilizes a permission-based access system to prevent applications from obtaining unauthorized access to sensitive resources, such as cameras, microphones, and internal file storage Permissions on Android. (2024). The Android Market Security Model functions similarly to the Linux security model, where permissions are based on user consent. A user cannot read, modify, or execute another user's files without explicit permission.

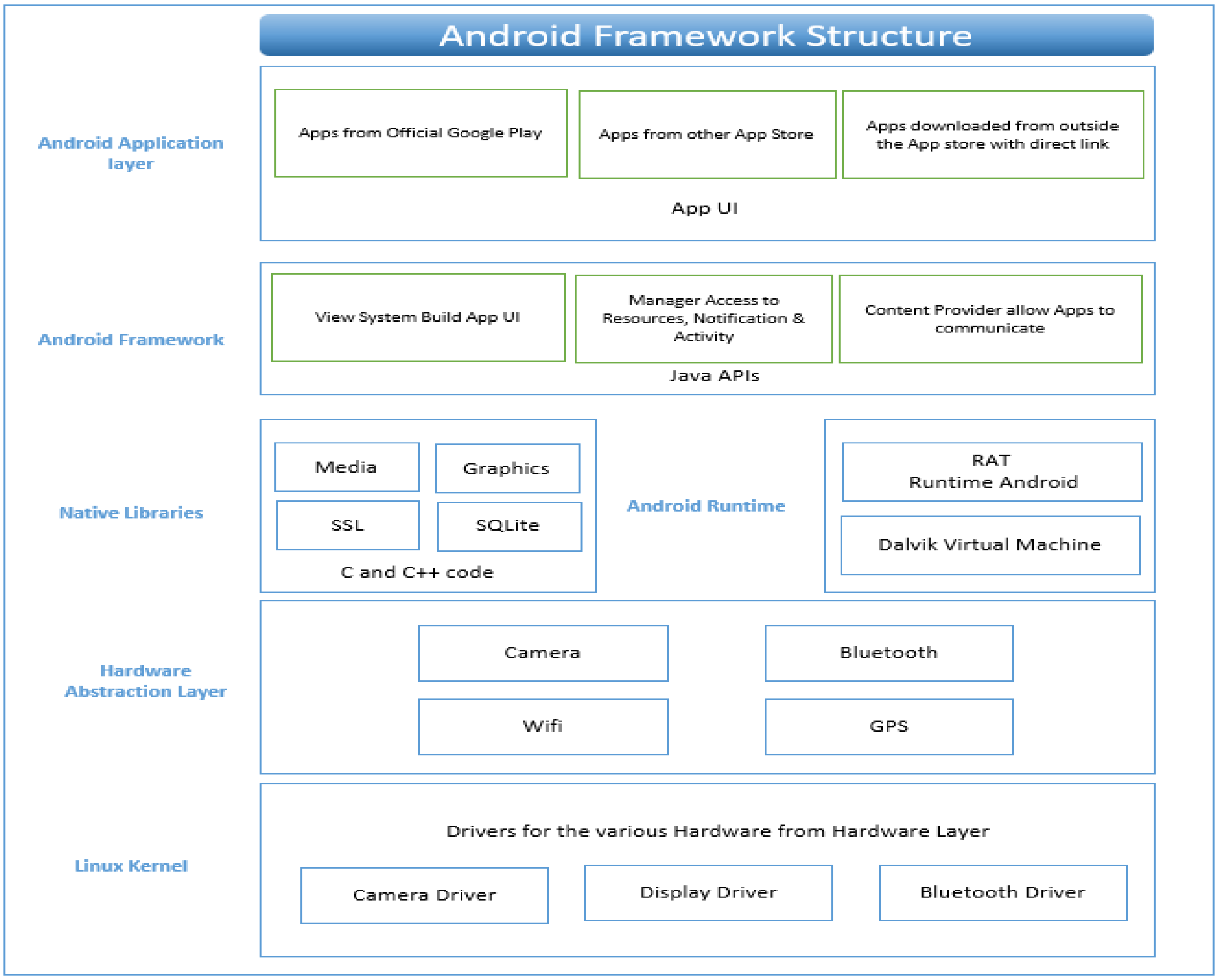
When applications are installed, they must request permissions from the user, which are based on the resources they intend to use and the areas they need to access. These permissions are outlined in the AndroidManifest.xml file within the APK (Android application package) Marko M. (2019, November 15). Malicious software, such as Trojans, ransomware, spyware, and worms, frequently exploits users who are unaware of Android's permission system, thereby jeopardizing their data. This underscores the need for user education regarding Android permissions. In the security model of the Android operating system, the responsibility lies with the individual installing the app to consciously grant these permissions. Third-generation app developers may use these permissions either intentionally or unintentionally. Ultimately, it is the user's duty-the person installing the app-to assess whether the requested permissions are essential for the app's functionality and to grant them accordingly.

Android Architecture

The Android operating system is organized into six main layers: The Application layer, Application Framework, Libraries, Android Runtime, Hardware Abstraction Layer, and the Linux Kernel (Figure 1). Each of these layers is essential to the overall functionality and security of the system.

Android Application Layer

The Android application layer allows developers to utilize the device's existing functionalities, such as accessing hardware features (like the camera, GPS, and sensors)


Android Framework Structure
Android Application layer
Apps from Official Google Play
Apps from other App Store
Apps downloaded from outside the App store with direct link
App UI
Android Framework
View System Build App UI
Manager Access to Resources, Notification & Activity
Content Provider allow Apps to communicate
Java APIs
Native Libraries
Media
Graphics
SSL
SQLite
C and C++ code
Android Runtime
RAT Runtime Android
Dalvik Virtual Machine
Hardware Abstraction Layer
Camera
Bluetooth
Wifi
GPS
Linux Kernel
Drivers for the various Hardware from Hardware Layer
Camera Driver
Display Driver
Bluetooth Driver

and integrating with system services (such as notifications, data storage, and network access). This capability enables them to create applications that interact seamlessly with the device's features.

Figure 1:
Android Platform Architecture

APPLICATION FRAMEWORK

The application framework is frequently used by developers and provides a range of higher-level services to applications through Java classes. These services enable developers to manage user interfaces, resources, and system functions more effectively, thereby simplifying the app development process.

Native Libraries

Android includes an open-source Web browser engine based on the open source WebKit, a well-established database for data storage; SQLite (a lightweight relational database management system) and support for many other standard c libraries. Libc and Libutil are system libraries, they provide essential functions of a given system. The native libraries are the core building blocks of android exposed with C, C ++ by providing performance — critical capabilities that serve as a base for the running system runtime and higher-level frameworks.

Android Runtime

The Android Runtime (ART) is like the third section, which comes under second layer from bottom in Android architecture. Down here is the crucial layer which houses Dalvik Virtual Machine (if you are still using it) or Android Runtime (ART, in case your version of Android). They facilitate execution of Android application code in an efficient manner, by translating app's Java-based code into system interpretable machine code intended for the processor. Last but not least, the Android Runtime manages memory and process lifecycle to ensure a bug free experience of running the apps.

Hardware Abstraction Layer

The HAL provides a standard API for creating software that is compatible with the hardware and Android API framework to exchange control data with regard to camera, Bluetooth, audio, sensors etc. HAL gives developers access to hardware features with a standardized interface, abstracting low-level hardware details from the application and making applications work consistently across multiple devices.

Linux Kernel

The kernel of the Android operating system is derived right from the open-source Linux Kernel, that sits at the core part. Functions such as memory management, process management, and device drivers are all controlled by the kernel. Similarly, also at the kernel level, a number of userspace services and libraries interact with these HALs facilitating communication between the Android system's hardware aspects and their software types. Improving stability, performance, and security from devices to the edge.

Androidmanifest.XML FILE

Every Android application has to have an AndroidManifest. This file contains metadata emulated in xml format, which tells the system about all the application components such as activities, services, broadcast receivers and content providers. These components are not executable unless declared in the manifest. Usually it contains what kind of device needs a camera, heart rate sensor or GPS etc. Also, the app needs to get permission from the user for accessing data that is protected by default (such as addresses, cameras, or other location information) This item is telling to AndroidManifest that I need these required permission. xml file. In a nutshell, this file is used for informing the operating system about the application properties and specifics so that it can keep track of its details and manage interactions. The primary components typically found in the manifest file include:

Package Name: This serves as a unique identifier for the application, distinguishing it from others.

Minimum and Maximum API Levels: Specifies the range of API versions with which the software can interact.

Component Description: Details the activities, services, broadcast receivers, and content providers utilized by the application.

Library List: Enumerates the libraries that the application will rely on.

Permission Declarations: Outlines the permissions required by the application to access and interact with specific components.

Common Types of Malicious Code on Android

Malicious code targeting Android devices can take various forms, each designed to exploit vulnerabilities and compromise user security. Understanding these common threats is crucial for users and developers alike to implement effective security measures and protect against potential attacks. Common types include:

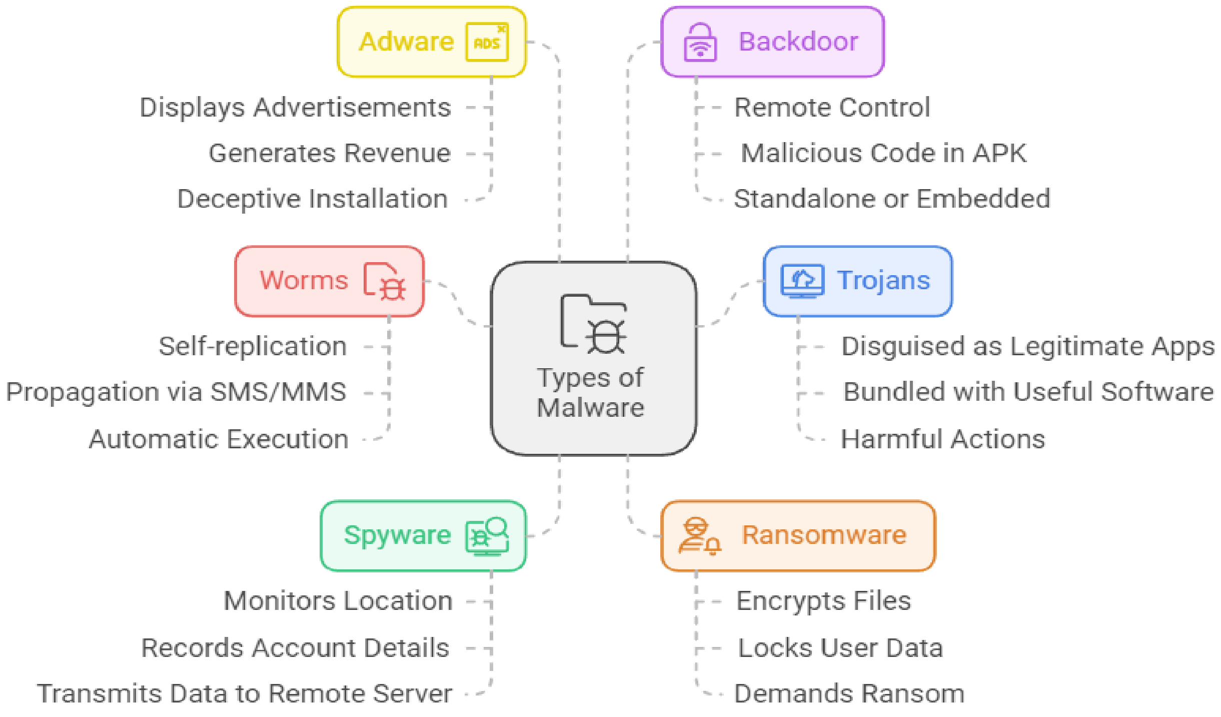


Figure 2:
Types of android Malware

Worms: Worms are a type of malware that can replicate themselves and transfer files to other devices. On Android devices, these worms are typically disseminated through SMS or MMS text messages and often execute automatically, without requiring any user intervention.

Trojans: Trojans are deceptive software programs that often disguise themselves as legitimate applications. They may be bundled with useful software but are designed to perform harmful actions, such as collecting personal information and stealing account credentials.

Spyware: Spyware is a type of software that gathers and transmits personal information without the user's consent. This software monitors users by recording their location, account details, and other sensitive information, sending it to a remote server. Many spyware programs are bundled with seemingly harmless applications, operating quietly in the background and making them challenging for users to detect.

Ransomware: Ransomware is a type of malware that is designed to lock users out of files on their computers, images, videos and even compressed or archive file format. In cases like these, users would have to pay a ransom if they want to recover their access back. Also ransomware can lock the device itself, making it an expensive ruse to reinstate access to the user.

Adware: Adware is a type of malicious software designed to display advertisements on the user's screen, generating revenue through ad views. Often masquerading as useful software or bundled with legitimate applications, adware aims to deceive users into installing it on their devices.

Backdoor: A backdoor is a type of malware that enables an attacker to remotely control a device, functioning as if a legitimate user were operating it. A backdoor can exist as standalone software or as part of a legitimate application, with malicious code embedded within an existing APK file.

Permissions in Android: Permissions are a crucial aspect of the Android operating system, with approximately 250 different types available Zhou, Y., & Jiang, X. (2012). They play a key role in defining the security measures and overall safety of applications. These permissions are categorized into four protection levels: Normal, Dangerous, Signature, System permission or Privileged Sihag, V. et.al (2021). .

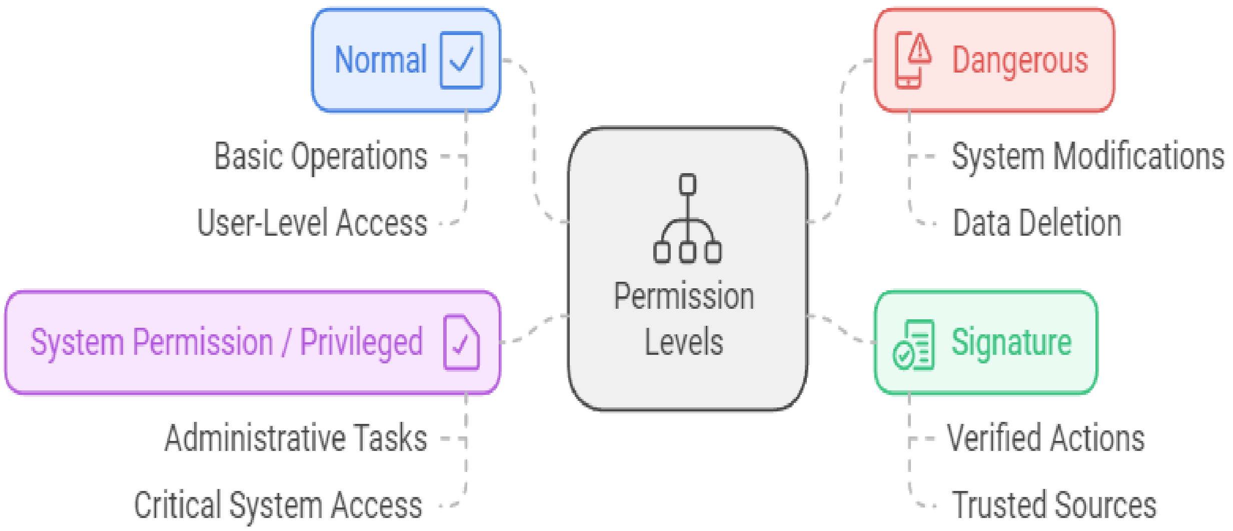


Figure 3:
Android Permission Levels

Among these, permissions categorized as 'Dangerous' are particularly significant because they manage users' personal data. If misused with malicious intent, they can jeopardize users' security and privacy. Therefore, obtaining user consent is mandatory for these permissions Zhou, Y., & Jiang, X. (2012). For instance, the SEND_SMS permission is essential for communication and social media applications that facilitate the sending of text messages. In essence, permissions represent the requests that applications make to the Android system to gain access to specific features and controls. For instance, if an application wants to access the internet, it must request permission from the system to do so. Typically, applications require permissions for Internet access, camera usage, and the ability to enable WiFi. These permissions are categorized into two levels: normal and dangerous. Normal permissions pose minimal risk to user privacy, while dangerous permissions involve access to sensitive data or system features that could significantly impact user security. Apps are required to declare their permission requests in their manifest file However, during the download and installation process, users are presented with the option to either accept or deny all permissions simultaneously. If a user declines the permission request, the app cannot be installed. Consequently, many users often accept all permissions without fully comprehending their implications or the associated risks. Additionally, apps can also request further dangerous permissions from users at runtime.

Normal Level Permissions

These permissions present minimal risk to user privacy and have little to no impact on the system's overall functionality. The Android operating system typically grants these permissions automatically to applications listed in the Android Manifest file, without requiring explicit user consent. Common examples of normal permissions for API version 23 are detailed in Table 1.

Table 1:
Normal Permission List

No	Permission
1	INTERNET
2	ACCESS_NETWORK_STATE
3	ACCESS_WIFI_STATE
4	BLUETOOTH
5	CHANGE_NETWORK_STATE
6	CHANGE_WIFI_STATE
7	EXPAND_STATUS_BAR
8	READ_SYNC_SETTINGS WILL
9	SET_ALARM
10	VIBRATE
11	WAKE_LOCK
12	WRITE_SYNC_SETTINGS
13	GET_PACKAGE_SIZE
14	RECEIVE_BOOT_COMPLETED
15	SET_ALARM

Dangerous Level Permissions

These permissions pertain to user privacy and can impact other applications or the functioning of the operating system. Unlike normal permissions, dangerous permissions are not granted automatically; the user must explicitly approve them when an application requests access. Examples of dangerous permissions are outlined in TABLE 2.

Table 2:
Dangerous Permission List

No	Permission
1	READ_CALENDAR
2	WRITE_CALENDAR
3	CAMERA
4	READ_CONTACTS WILL
5	WRITE_CONTACTS
6	GET_ACCOUNTS
7	ACCESS_FINE_LOCATION
8	ACCESS_COARSE_LOCATION
9	RECORD_AUDIO

10	CALL_PHONE
11	READ_CALL_LOG
12	WRITE_CALL_LOG
13	SEND_SMS
14	RECEIVE_SMS
15	READ_SMS

Malware can take advantage of this permission to communicate with command centers or send messages to premium-rate numbers, which may lead to unexpected charges. While granting permissions individually might appear harmless, allowing them together can drastically increase privacy and security risks. For example, the INTERNET and READ SMS permissions, when granted separately, pose minimal threat. However, when combined, they can enable an app to access your text messages and send them to an external entity Sihag, V. et.al (2021). Whether permissions are considered normal or dangerous, they must be specified in the Android Manifest file. This declaration notifies the Android operating system of the permissions the application needs to function correctly.

Librarys in Apk Files

Libraries within an APK (Android Package Kit) file are software components that enable Android apps to perform specific functions without the need to write code from the ground up. These libraries can include:

System Libraries

These are libraries supplied by the Android SDK, including components like android.app, android.content, and android.view. They deliver essential functionalities that are fundamental to the Android operating system.

Third Party Libraries: These are libraries created by developers that are not included in the Android SDK. Examples include:

Retrofit: A library for managing HTTP requests.

Glide or Picasso: Libraries for loading and processing images.

Room: A library for handling SQLite database management.

Open Source Libraries: These libraries are freely available and open source, allowing developers to incorporate common functionalities without the need to build them from scratch. In this study, we utilize the names of third-party libraries as identifiers.

Android Vulnerabilities

Android vulnerabilities refer to weaknesses within the operating system or its applications that can be exploited by malicious actors.

Fragmentation Problem: Typically, Google releases an Android update each month, but it may take several months for these updates to be distributed to users across different manufacturers worldwide. This delay leads to the presence of multiple Android versions in use globally, with older versions remaining susceptible to security vulnerabilities that have been fixed in later updates Bagheri, H. et.al (2017). . For instance, some permissions deemed dangerous in the most recent update may still be regarded as normal on devices that have not received the update, thereby heightening the risk of user data exploitation Faruki, P. et.al (2016).

Colluding Attack: When users inadvertently install several applications that are signed with the same developer certificate, those applications are able to share permissions and access resources. Although each app may request permissions that appear harmless, the cumulative effect of these permissions can enable malicious activities. Furthermore, each app can independently access various resources. Since all applications associated with the same certificate can leverage these resources, one app could collect a considerable amount of data without triggering any alerts [15, 21].

Malware Functionalities

Malware serves different purposes and employs various attack methods. Some types disrupt device functionality by launching extensive ad attacks, while others may steal users' contacts to spread malicious activity and target additional victims. More harmful variants can impose financial charges on users or even steal bank account information to carry out unauthorized transactions. Below, we outline the functionalities of these malware types.

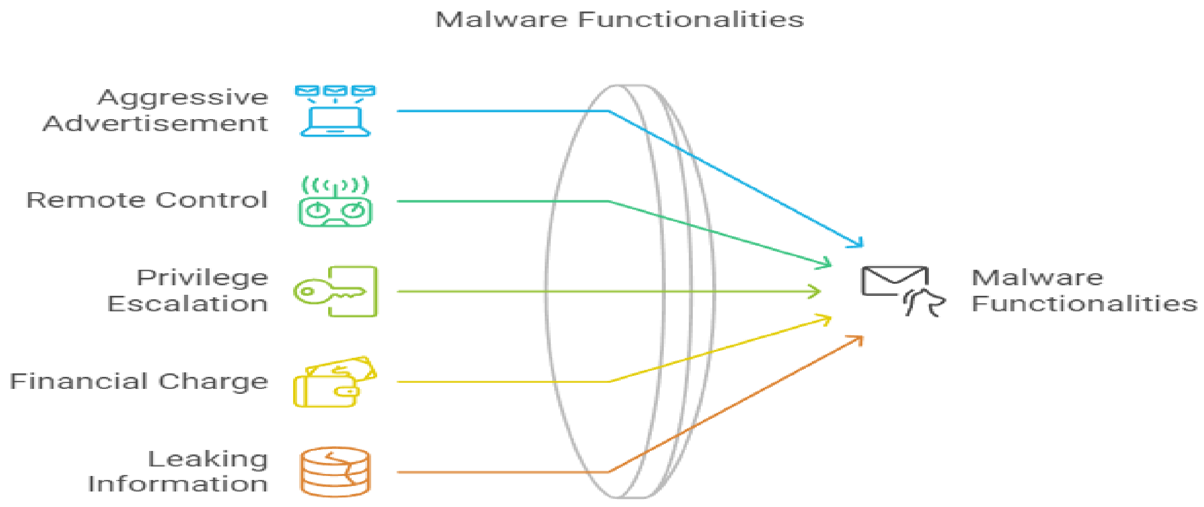


Figure 4:
Malware Functionalities

Aggressive Advertisement

A common form of malware encountered by users involves bothersome applications that produce constant pop-ups, hindering the device's functionality. Certain malware can seize control of the user's device, overwhelming them with ads, changing their default search engine, and executing other intrusive behaviors, as seen with the Plankton malware Thomas, D. et.al (2015). .

Remote Control

About 93% of malware employs compromised devices as bots Bagheri, H. et.al (2017). . By infiltrating a device, this malware can seize control and incorporate it into a botnet-a collection of devices overseen by a remote server-either to steal information or carry out attacks, such as denial of service assaults. Notable examples of malware that create botnets include Beanbot and Anserverbot Thomas, D. et.al (2015). . Beanbot targets devices to steal information, such as the IMEI number and phone number, sending this data to a remote server. It can also transmit expensive SMS messages from the device, depleting phone credits. Anserverbot embeds code onto the victim's device, granting the hacker remote access. This malware often hides within an app and prompts users to install what appears to be an update; in reality, this action downloads and installs the remote control program on the victim's device.

Privilege and Permission Escalation

Approximately 36% of malware utilizes at least one root exploit, although it is common for multiple exploits to be employed simultaneously. By taking advantage of the Android vulnerability colluding attack, various forms of malware can work together to share permissions and gain elevated privileges Arzt, S. et.al (2014). . Notably, permissions related to SMS are the most commonly targeted; around 45% of malware seeks access to various SMS functions, such as reading, writing, receiving, and sending messages Faruki, P. et.al (2015). Malicious software exploits these heightened permissions for a range of purposes. Additionally, it has been noted that harmful apps generally request more permissions than legitimate ones. On average, malicious applications seek around 11 permissions, whereas benign apps typically request about four permissions.

Financial Charge

Certain types of malware leverage remote control and privilege escalation to exploit compromised devices for financial gain. This can involve sending messages or subscribing to premium-rate numbers, messaging contacts, or even making phone calls in the background without the user's awareness Faruki, P. et.al (2015). For instance, DroidSMS can subscribe users to premium numbers, while Zitmo is designed to capture login information to facilitate unauthorized financial transactions from users' bank accounts Zhou, Y., & Jiang, X. (2012). . A specific type of malware, known as Zitmo, poses a significant threat to mobile users by targeting sensitive transaction authorization codes. This Trojan horse malware intercepts and forwards incoming text messages containing these codes to unauthorized parties, enabling them to carry out fraudulent financial transactions using compromised accounts. Another malicious software variant that results in financial losses is ransomware, which restricts device access until the user pays a ransom to recover their data. This type of malware has been observed in various forms, including FakeDefender, which exemplifies the malicious tactics employed by cybercriminals to extort money from unsuspecting victims.

Leaking Information

Applications typically require access to user information to operate and facilitate communication. However, transmitting this data beyond the user's device without their awareness or consent constitutes information leakage Arzt, S. et.al (2014). . Over 80% of malware programs gather personal and device-related details and transmit them to external servers. These details may include device identifiers like IMEI, IMSI, kernel version, phone manufacturer, and network operator information Salehi et.al (2019). . Moreover, malware has the capability to gather SMS messages, passwords, user accounts, usernames, email addresses and phone numbers. With such data, malicious actors can commit fraud against users without their knowledge. For instance, the malware FakeNetflix impersonates the legitimate Netflix app to steal user login credentials and leak them Faruki, P. et.al (2015).

Malware Analysis Techniques

To successfully track, identify, and combat malware activities, a lot of research has been done over the years. This has led to the development of various anti-malware methods, each using different ways to detect threats. In the following sections, we will explore both static and dynamic anti-malware techniques, along with cutting-edge tools utilized in each approach.

STATIC ANALYSIS

Static analysis is a common approach for finding malware by examining and breaking down Dalvik bytecode without running the code. This approach helps to prevent malware from disguising or stalling the execution of its actions during the analysis. CHEX or FlowDroid are such tools that have been developed to scan through the code to detect any potential vulnerabilities in the app. These vulnerabilities may victimize you as service provider by leaking some of data back (data leak), or escalating a permission to access, execute some functions unintentionally. While FlowDroid is employed for the analysis of sensitive data flows, CHEX is utilized to keep a track of which entry points or potential routes may lead to such hijacking vulnerabilities; however, both these techniques perform good only up-to certain extent and some limitations like how they go through implicit flows into reflective calls. Expanding on these, techniques in Amandroid and Apposcopy help to identify some more advanced sensitive data leaks.

Amandroid finds misuse of APIs and maps data flows between app components well, but does not handle concurrency and implicit flows properly. BitAPot detects the malware family regardless of code obfuscation but not in variants unknown to the predefined behavior by using signature-based analysis, and apposcopy is a complementary approach. Tools like EviHunter are for forensic examination, allowing the user to build a database of evidentiary data so they can compare rival apps. These tools can then help you stay out of possible patent infringement. However, as with the tools from before they have issues discovering dynamic payloads that appear only at runtime. Static analysis, despite several approaches such as SafeDroid and AnaDroid and techniques based on power consumption to detect dynamic behaviors easily evades it.

DYNAMIC ANALYSIS

Dynamic analysis is another effective technique for detecting and mitigating malware by executing and monitoring app behavior in controlled environments like emulators, simulators, or sandboxes. This method simulates user interactions to observe app functionality, control-flow, and actions, distinguishing between malicious and benign behavior. Tools such as ServiceMonitor analyze app interactions with system services, using statistical models like Markov chains and Random Forest algorithms to classify apps based on their behavior. Although this technique can detect many types of malware, some evade detection by recognizing the analysis environment, leading to false negatives. TaintDroid and TaintART track sensitive data as it flows through the system, flagging apps that transmit data outside the device. TaintDroid focuses on explicit data flows, but struggles with implicit flows, a limitation also seen in TaintART. Similarly, Droid-AntiRM improves dynamic analysis by addressing anti-analysis techniques, forcing malicious behavior to be executed during testing. However, it still cannot handle dynamic code loading or obfuscation methods used by sophisticated malware Detection of Intrusions and Malware, and Vulnerability Assessment. (2017). Other tools, such as DroidScope and several dynamic analysis frameworks like DL-Droid, CopperDroid, and MAdFraud, attempt to tackle malware detection at different levels. However, they face a common limitation:

many malware programs detect the analysis environment and either avoid triggering their malicious actions or crash to prevent further investigation. This vulnerability highlights the challenge dynamic analysis faces when dealing with advanced malware tactics designed to bypass detection.

HYBRID ANALYSIS

Hybrid analysis techniques combine static and dynamic methods to improve malware detection. A tool like WifiLeaks uses static analysis to find what permissions an app asks for, and then application of dynamic analysis to monitor the use of these permissions in data collections and possible data leaks. WifiLeaks is a research tool designed for the detection of security degradation in ongoing WiFi access permission, any Android app can be classified (a malicious one or not) by using Dalvik bytecode get from apk file. EspyDroid focuses on malware using reflection and obfuscation, we use static analysis to further reduce the code path for better dynamic analysis. AndroShield is another hybrid solution, which performs static reverse engineering of APK files to study the code and manifest files. Then, during dynamic analysis, the app is run and monitored for behavior which could result in data leaks (e.g. logging sensitive information), crashes, requests being sent insecurely etc. The other types of hybrid tools such as SamaDroid, AspectDroid and AndroPyTool mix both of the inspection and behavior extraction, however, they face a few obstacles in the way as dealing with obfuscation or package signature changes or their defense mechanisms are not liable to zero-day attacks. Hybrid analysis combines the advantages of both casual but as discussed, which faces challenges that static and dynamic analysis do for example advanced obfuscation or some new brewed malware detection. This has fueled a need for increasingly sophisticated techniques that can perform modifications and in turn adapt without being centered around innocent or familiar behavior patterns that other attacks use.

ANTI MALWARE EVASION TECHNIQUES

With the continual propagation of malware, various anti-malware techniques have become available for treating malware threats. However, malware creators are constantly evolving and refining their methods to bypass these detection strategies. Here are some of the most common methods used by malware developers to avoid being detected by anti-malware software.

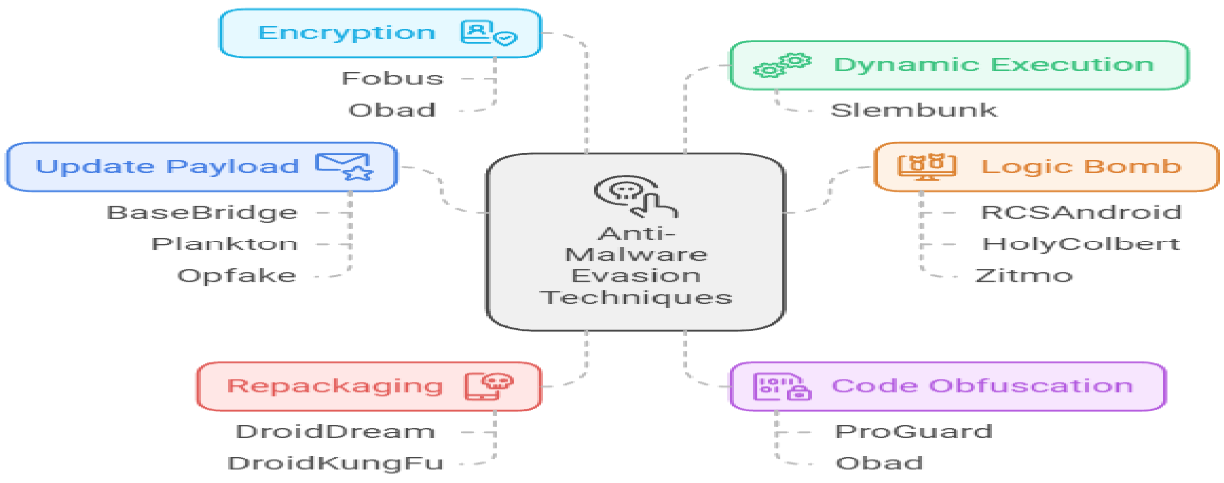


Figure 5:
Anti-Malware Evasion Techniques

Repackaging

Malware developers frequently use reverse engineering to compromise legitimate Android apps. They start by downloading a popular app, adding harmful code to it, and then rebuilding the app. The infected version is then republished on official or third-party app stores. When users unintentionally install the app, they become open to malware attacks that can take their personal information or make unauthorized purchases. This repackaging technique is highly prevalent, with over 85% of malware, such as DroidDream and DroidKungFu variants, employing this method.

Update Payload

Another method malware developers use to evade anti-malware tools is called an update attack or dynamic payload. Instead of putting all the harmful code directly into the initial app, they attach the malicious code as an APK/JAR file and encourage users to install important updates. These updates then download the harmful code from a remote server. This method evades signature-based and static scanning tools. Malware families like BaseBridge and Plankton commonly use this technique. Additionally, some malware, like Opfake, employs polymorphism to alter its code with each update without changing its functionality. Polymorphism allows malicious code to exploit the same methods by overriding behavior through inheritance, making it harder to detect.

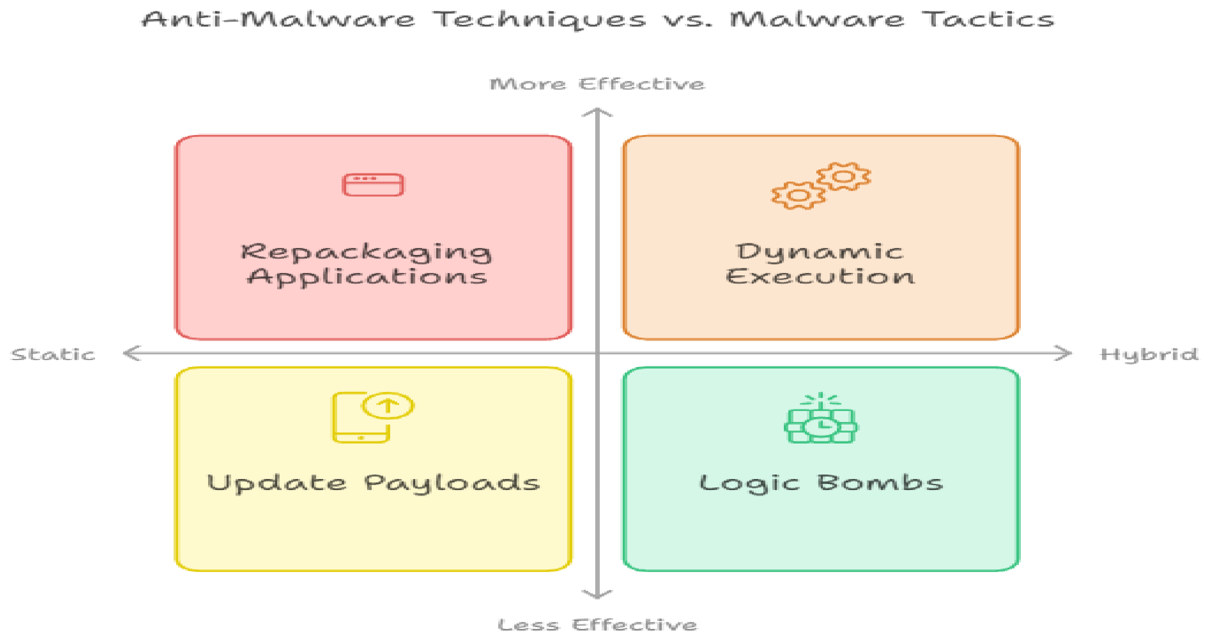


Figure 6:
Anti-Malware Techniques Vs Malware Tactics

Dynamic Execution

Malware frequently registers to listen for system-wide events such as Boot, Call, and SMS. When these events happen, the malware activates its harmful code. For instance, Slembunk uses this method to track user activities, and when the user opens their banking app, it shows a fake screen that looks like the real app. This enables Slembunk to steal the user's banking details straight from the infected device.

Code Obfuscation

App developers frequently use code obfuscation methods to protect their intellectual property from being misused and to make reverse engineering more difficult. This not only defends their code but also yields a much smaller app that users can run smoothly on their devices. One of them is an optimization tool used in build process like ProGuard, it optimizes your app code by removing any unused classes and methods and also shortening class names to obfuscated names. Still, these same techniques have been adopted by malware developers as a way to evade detection. Other obfuscation technique can be used like adding junk code, renaming packages or controlling the flow of the program. For instance, the Obad malware obfuscates every class and method name with unreadable strings which makes it almost invisible to the service. And it is particularly elusive when you can't see them in the list of device administrators, from root access. In order to prevent those advanced malware, users can disable the auto-discovery options in Android operating system and to scan their devices with reliable antivirus software for potential threats as well.

Encryption

Another way to avoid detection is by encrypting code that is only decrypted when the app is running. Different encryption methods, like string or class encryption, can make apps more secure. To analyze the malware, researchers need to decrypt it and convert the encrypted text back to regular text to understand how it works. However, finding the encryption key can be quite difficult. For instance, Fobus uses the name of the fourth class and method as a key for the JVM stack, while Obad creates its key from a particular Facebook page.

Logic Bomb

Some malware can avoid both dynamic and static analysis by not activating their harmful code when they are run. Instead, they wait for specific events to trigger their harmful actions, as seen with RCSAndroid. Others may employ a delay before activating their malicious code, a tactic known as a time bomb, exemplified by HolyColbert. Furthermore, some malware displays a login screen that asks for user credentials to continue, which effectively prevents analysis tools from examining the app's functions and actions, as seen with Zitmo.

CONCLUSIONS AND FUTURE WORK

The study explored the key vulnerabilities in Android, how it is manipulated by malware developers, and different types of attacks implemented using these weaknesses. These attacks can range from a simple popup-ad to remote device control, through heavy frauds, misusing of data privacy protocols and also both online scamming-which usually ends up being an isolated case- but sometimes ripping out as unauthorized privilege escalation affecting several other users – causing a never-ending series of data breach. Based on extensive literature, we have provided an overview of different types anti-malware techniques and categorized them in to static, dynamic and hybrid methods while analyzing the effectiveness of these approaches with respect to particular attack types as well as datasets utilized during performance evaluation. We also considered some common techniques malware developers turn to in order to bypass these detection methods, like repackaging apps, rolling update payloads and dynamic execution, code obfuscation and logic bombs. Future work will explore reinforcement learning techniques to address sustainability issues suffered by current anti-malware solutions. The high level approach for us is to use some machine learning techniques in order to build a more flexible and durable system that can adapt based on the strategy used by malware developers. To these API s (we will cover APIs and parameter use cases in more depth later) the security layer could be added while executing security functionality on execution level, a single step approach to greatly enhance anti malware tools efficiency in protecting users from new or emerging threats.

DECLARATIONS

Acknowledgement: We appreciate the generous support from all the supervisors and their different affiliations.
Funding: No funding body in the public, private, or nonprofit sectors provided a particular grant for this research.
Availability of data and material: In the approach, the data sources for the variables are stated.
Authors' contributions: Each author participated equally to the creation of this work.
Conflicts of Interests: The authors declare no conflict of interest.
Consent to Participate: Yes
Consent for publication and Ethical approval: Because this study does not include human or animal data, ethical approval is not required for publication. All authors have given their consent.

REFERENCES

Android TV. (n.d.). Android. https://www.android.com/tv/
Marko M. (2019, November 15). Android vs iOS Market Share Discoveries in 2023. Leftronic.com; Leftronic. https://leftronic.com/blog/android-vs-ios-market-share
Nick Jasuja. (2019). Android Vs iOS - Difference and Comparison | Diffen. Diffen.com; Diffen. https://www.diffen.com/difference/Android_vs_iOS
(2024). Av-Test.org. https://www.av-test.org.
Turner, A. (2022, January 12). How Many Android Users Are There? Global Statistics (2023). BankMyCell. https://www.bankmycell.com/blog/how-many-android-users-are-there
Wang, W., Li, Y., Wang, X., Liu, J., & Zhang, X. (2018). Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers. Future Generation Computer Systems, 78, 987–994. https://doi.org/10.1016/j.future.2017.01.019
Kivva, A. (2023, June 7). IT threat evolution Q1 2023. Mobile statistics. Securelist.com; Kaspersky. https://securelist.com/it-threat-evolutionq1-2023-mobile-statistics/109893
Mathur, A., Podila, L. M., Kulkarni, K., Niyaz, Q., & Javaid, A. Y. (2021). NATICUSdroid: A malware detection framework for Android using native and custom permissions. Journal of Information Security and Applications, 58, 102696. https://doi.org/10.1016/j.jisa.2020.102696
Mathur, A., Ewoldt, E., Quamar Niyaz, Javaid, A. Y., & Yang, X. (2022). Permission-Educator: App for Educating Users About Android Permissions. Lecture Notes in Computer Science, 361–371. https://doi.org/10.1007/978-3-030-98404-5_34
Permissions on Android. (2024). Android Developers. https://developer.android.com/guide/topics/permissions
Shi, S., Tian, S., Wang, B., Zhou, T., & Chen, G. (2023). SFCGDroid: android malware detection based on sensitive function call graph. International Journal of Information Security. https://doi.org/10.1007/s10207-023-00679-x

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., Yinwei, W., & Jiaxuan, G. (2021). A Systematic Overview of Android Malware Detection. Applied Artificial Intelligence, 1–33. https://doi.org/10.1080/08839514.2021.2007327

Zhou, Y., & Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. 2012 IEEE Symposium on Security and Privacy. https://doi.org/10.1109/sp.2012.16

Sihag, V., Vardhan, M., & Singh, P. (2021). A survey of android application and malware hardening. Computer Science Review, 39, 100365. https://doi.org/10.1016/j.cosrev.2021.100365

Bagheri, H., Kang, E., Malek, S., & Jackson, D. (2017). A formal approach for detection of security flaws in the android permission system. Formal Aspects of Computing, 30(5), 525–544. https://doi.org/10.1007/s00165-017-0445-z

Faruki, P., Fereidooni, H., Laxmi, V., Conti, M., & Gaur, M. (2016). Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions. ArXiv.org. https://arxiv.org/abs/1611.10231

Thomas, D. R., Beresford, A. R., & Rice, A. (2015). Security Metrics for the Android Ecosystem. Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices. https://doi.org/10.1145/2808117.2808118

Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., & McDaniel, P. (2014). FlowDroid. ACM SIGPLAN Notices, 49(6), 259–269. https://doi.org/10.1145/2666356.2594299

Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2015). Android Security: A Survey of Issues, Malware Penetration, and Defenses. IEEE Communications Surveys Tutorials, 17(2), 998–1022. https://doi.org/10.1109/COMST.2014.2386139

Bhandari, S., Jaballah, W. B., Jain, V., Laxmi, V., Zemmari, A., Gaur, M. S., Mosbah, M., & Conti, M. (2017). Android inter-app communication threats and detection techniques. Computers & Security, 70, 392–421. https://doi.org/10.1016/j.cose.2017.07.002

Zhou, Y., & Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. 2012 IEEE Symposium on Security and Privacy. https://doi.org/10.1109/sp.2012.16

Detection of Intrusions and Malware, and Vulnerability Assessment. (2017). In M. Polychronakis & M. Meier (Eds.), Lecture Notes in Computer Science. Springer International Publishing. https://doi.org/10.1007/978-3-319-60876-1

Salehi, M., Amini, M., & Crispo, B. (2019). Detecting malicious applications using system services request behavior. Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services. https://doi.org/10.1145/3360774.3360805

2024 by the authors; EuoAsian Academy of Global Learning and Education Ltd. Pakistan. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).