IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.Doi Number

Protecting Android Devices from Malware Attacks: A State-of-the-Art Report of Concepts, Modern Learning Models and Challenges

ESRA CALIK BAYAZIT[1], OZGUR KORAY SAHINGOZ[2], AND BUKET DOGAN[3]

[1]Computer Engineering Department, Fatih Sultan Mehmet Vakif University, Beyoglu, Istanbul, 34421 Turkey
[2] Computer Engineering Department, Biruni University, Topkapi, Istanbul, 34093, Tukey
[3]Department of Computer Engineering Faculty of Technology, Marmara University, Aydinevler, Istanbul, 34854Turkey

Corresponding author: Esra Calik Bayazit (e-mail: ecalik@ fsm.edu.tr).

"This work was supported in part by Marmara University Scientific Research Projects Coordination Unit under grant number FDK-2020-10066."

ABSTRACT Advancements in microelectronics have increased the popularity of mobile devices like cellphones, tablets, e-readers, and PDAs. Android, with its open-source platform, broad device support, customizability, and integration with the Google ecosystem, has become the leading operating system for mobile devices. While Android's openness brings benefits, it has downsides like a lack of official support, fragmentation, complexity, and security risks if not maintained. Malware exploits these vulnerabilities for unauthorized actions and data theft. To enhance device security, static and dynamic analysis techniques can be employed. However, current attackers are becoming increasingly sophisticated, and they are employing packaging, code obfuscation, and encryption techniques to evade detection models. Researchers prefer flexible artificial intelligence methods, particularly deep learning models, for detecting and classifying malware on Android systems. In this survey study, a detailed literature review was conducted to investigate and analyze how deep learning approaches have been applied to malware detection on Android systems. The study also provides an overview of the Android architecture, datasets used for deep learning-based detection, and open issues that will be studied in the future.

INDEX TERMS Android, Deep Learning, Malware Detection System, Malware Analysis, Machine Learning

I. INTRODUCTION

As smart mobile devices continue to capture global interest, manufacturers have responded to this demand by developing various devices that cater to different status groups. This has resulted in a remarkable shift from traditional computers to mobile devices, as evidenced in Fig. 1 [1]. The worldwide market share of mobile devices has surged tenfold due to increasing demand and technological advancements. According to the "Digital 2023" report, the number of smart mobile devices in use worldwide has reached a staggering 5.3 billion, with 71.63% of these devices running on the Android operating system [2]. Android's open-source architecture, accessibility, and scalability advantages have made it the most popular operating system for mobile devices, as shown in Fig. 2 [3]. Despite its popularity, the Android operating system has become a prime target for cyber attackers due to the valuable information it contains. Attackers use malicious applications that can be uploaded to Google Play, and particularly Android.

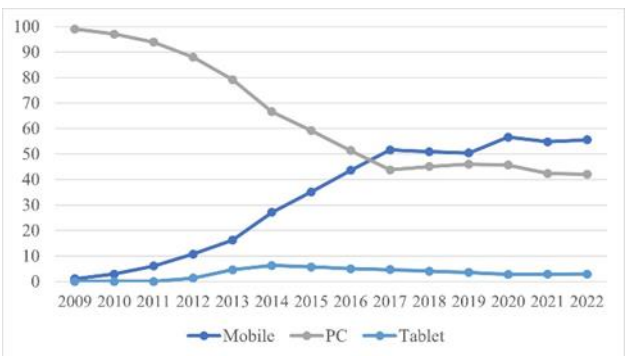

FIGURE 1. Worldwide market share of platforms.

applications that can be downloaded from third-party sources, to gain access to users' systems This has led to an increase in the number of applications on Android devices, making it

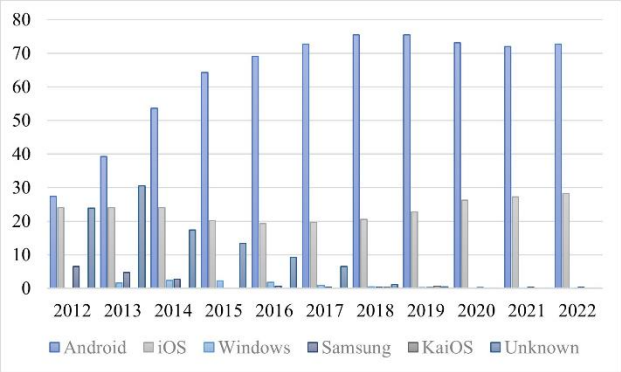

FIGURE 2. Market share of operating systems.

easier for malware to penetrate mobile devices, which, in turn, results in many security vulnerabilities. Cyber attackers can steal sensitive information from users, damage their devices, and send text messages to their contacts. In 2022, there were 3.48 million downloadable mobile applications, while the total number of Android malware was 3.36 million [4]. The number of Android malware is on the rise, and therefore, there is a need for malware detection and system analysis [5].

Current Android malware analysis approaches are categorized into static analysis, dynamic analysis, and hybrid analysis. Static analysis tries to detect Android malware by examining some files such as permissions, API calls, byte code extracted from the Android Manifest file of the application without running the application on the device. Therefore, its features are to analyze the semantic information of source and byte codes of the application package with reverse engineering methods [6]. Due to testing applications without running them, static analysis is safe and cost-effective. However, they are more vulnerable to code obfuscation and code polymorphism [7].

Dynamic analysis is a malware detection method that works by running malicious code in a real environment. The primary benefit of this method is that it identifies dynamic code loading and records the application's behavior during run time [8], [5]. It requires a virtual environment to monitor sensitive information during execution such as files, processes, connection requests, data flow, network traffic, processes. It monitors all suspicious requests and actions of malware, ensuring detection before it released into the network. This approach is time and resource costly but is an effective method for malware detection [9].

Static and dynamic analysis methods have some advantages and disadvantages compared to each other. For example, malware writers can pass static analysis with code obfuscation techniques, on the other hand dynamic analysis can easily catch it. On the other hand, dynamic analysis techniques are time costly and have less accuracy in the used classification model [9, 10]. Therefore, hybrid analysis which combines both static and dynamic analyses is preferred in many application domains. However, it also necessitates additional deficiencies in terms of feature engineering [11].

Artificial intelligence (AI) research has made significant progress in recent decades, and it has begun to shape our daily lives in every field of the cyber domain [12,13]. Various machine learning models, as a subset of artificial intelligence, can be used for adding intelligence to the implemented model [14]. Recent studies have shown that the focus is shifting from machine learning based systems to deep learning-based systems implementation due to their better efficiencies, especially with the use of larger datasets. Although in the literature there are many machines learning-based solutions for Android malware detection, in recent years deep learning based approaches have gained a huge importance and applied in some malware analysis implementations [15-17].

In this paper, it is aimed at providing an up-to-date and global survey of the Deep Learning-based Android Malware Detection implementations that have been published in recent years. With this work, not only researchers but also practitioners can find most of the background knowledge about the topic, which includes current Android malware datasets, malware analysis techniques, and deep learning algorithms. Although, in the literature, there are some surveys [8], [15-20] which examine different learning methods, it is seen that the studies are based on traditional machine learning models (RF, DT, SVM etc.) with older datasets.

In this survey, we examined more comprehensive research, which is separate from the former ones, with the following highlighting features:

- In this article, the Android architecture, new malware targeting Android systems, and malware analysis techniques are examined to provide an overview of Android systems and to create a framework for researchers.
- Current studies on deep learning-based malware detection in Android systems have been detailed.
- A comparative table is presented that includes the analysis methods, learning models, datasets used, characteristics of the dataset, and system performance results in the current literature studies examined.
- Deep learning models are defined by depicting their discriminative features.
- A comparative table of malware detection approaches is shown.
- A comparative table of current datasets is depicted.

The remainder of this article is organized as follows: Sections 2 and 3 discuss the Android overview and malware analysis that are fundamental in an efficient malware detection system, respectively. In Section 4, datasets used in the current Android malware detection systems in the literature are compared; the use of deep learning- based models in Section 5, current literature studies and their comparison table are presented in Section 6. Finally, clarifications for future work are drawn in Section 7 and conclusions are given in Section 8.

II. ANDROID OVERVIEW

This section provides an overview of the structures that make up Android systems and provides information on potential

threats to Android systems that are the subject of malware analysis. It also introduces the deep learning-based malware analysis process, which will help background readers understand the content of this article.

A. ANDROID APPLICATION PACKAGE

An Android Package Kit (APK) is an archive file in the Android system with the ".apk" extension that contains all the data and resource files needed to distribute and install applications to Android devices. Google Play serves as the primary store where users can search for and find their desired apps, movies, and shows, offering two million different apps and games to a vast number of users worldwide. To provide the best experience for specific devices, optimized APKs are created and delivered. Before uploading these apps to Google Play, the".apk" archive files are created and then added to the store in an executable format after undergoing necessary security checks. Although APK files can also be downloaded manually from third-party sources, caution should be exercised because these files contain all the necessary components, and downloading from unknown or unofficial sources can pose a security risk.

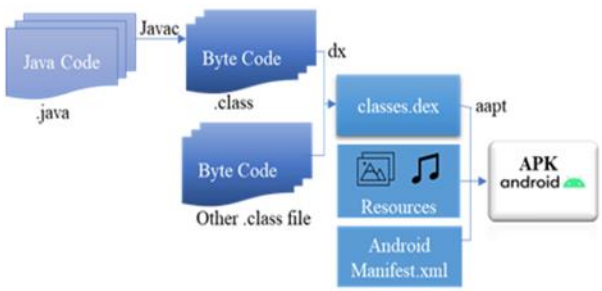

FIGURE 3. APK conversion steps.

Although Android is an open-source operating system, the source codes of system files are not readily accessible. However, it is possible to access all or part of the source codes using reverse engineering methods [21]. Once these codes are obtained, they can be modified and converted back into APK files. Typically, Android applications are written in Java, which is then compiled and converted into byte codes. Since Android cannot directly execute byte codes, they must first be converted into workable codes in the Dalvik virtual machine, which is the virtual machine that Android runs on. Once the APK conversion steps illustrated in Fig. 3 are completed, Android devices can run these codes.

B. APK FILE STRUCTURE

An APK file is a compressed file format that contains various components, including native libraries, resources, assets, certificates, and manifest files and directories. The most critical component of this compressed APK file is the "AndroidManifest.xml" file, where permissions for access and control are declared. Malware detection features are extracted from the APK file, which contains all the data and source files of the application, making the manifest file a crucial part of the process [22].

For an app to access resources, system files, and sensors like cameras, internet, contacts, and location, it must obtain necessary permissions. These permissions are categorized as normal and dangerous and are required by all applications. Additionally, an APK file must be signed by its creator, using the same certificate for updates to take effect. Certificates are crucial for ensuring the security of applications. Unfortunately, some unofficial providers may certify hundreds of developers' applications with the same signature, creating security vulnerabilities in various aspects such as the certificate signature. The Android file structure includes components such as "AndroidManifest.xml", Meta-Inf, assets, resources. arsc, lib, classes.dex, and res. Fig. 4 displays the structure of these components.

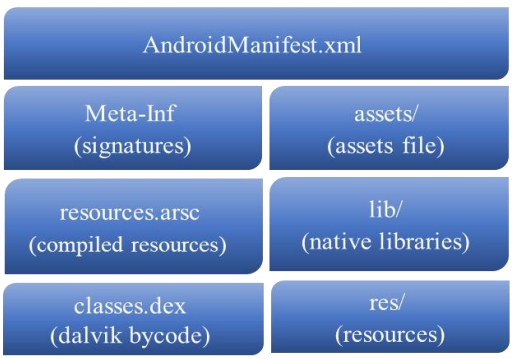

FIGURE 4. APK file structure.

AndroidManifest.xml The Android Manifest file contains the static analysis features of the applications, the package name of the application, the application per- missions, etc. XML file containing information. It is written in readable XML format and converted to binary XML during application execution. The AndroidManifest.xml file is the basic configuration file that configures the permissions and other parameters, used to guarantee the correct execution of applications. Permissions are a feature included in the Android Manifest file and are often used for malware detection. In this respect, permissions in Android systems act as a security mechanism that limits the direct access of applications to user sensitive data, as well as to resources and actions that are important to the system [23]

meta-inf — Directory containing APK metadata such as signature and certificate.

assets — It is the directory containing the files where the assets are kept.

resources.arsc — It is the file where XML files are compiled and put together. File with precompiled resources such as strings, colors, or styles.

lib— There are libraries (like armeabi, x86) needed by the application that are not included in the Android SDK.

classes.dex— It is a file containing application code in Dex file format, where Java codes are compiled into Dalvik Byte code.

res — The directory containing all uncompiled resources" resources. arsc" is all resources except files.

C. ANDROID SYSTEM ARCHITECTURE

The layers of the Android system, which has a layered system architecture, and the components they contain are presented in Fig. 5. Android has a customized Linux operating system (Linux Kernel) in its lowest layer that communicates with the phone hardware. The libraries layer on the Linux kernel includes the Web browser engine WebKit, Libc, SQLite database, a repository for storing and sharing application data, audio and video playback and recording libraries, SSL, internet security. In this category, there are Java-based libraries offered for application development [24].

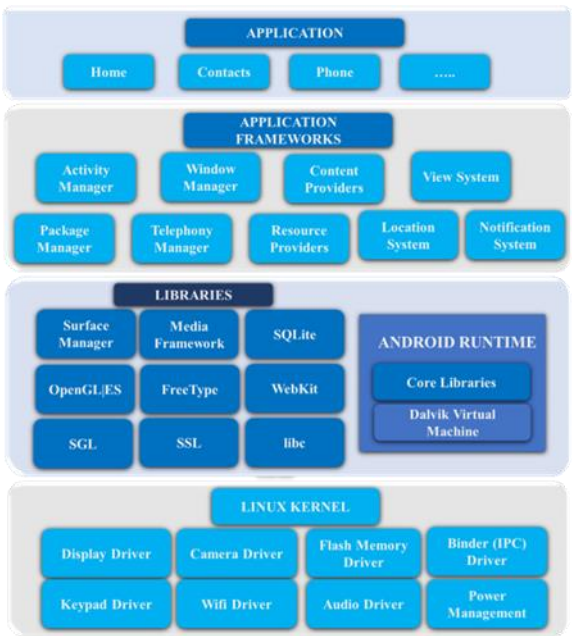

FIGURE 5. Android system architecture.

Examples of some core libraries are:

- android.content which enables messaging between applications and their components.
- android.opengl provides a Java interface to the graphics rendering API,
- android.widget contains a collection of UI components such as buttons, labels, list views,
- android.text renders text-based functions on the screen,
- android.webkit- allows the use of Web browser features in applications,
- android.os provides access to standard operating system services,
- android.view is one of the libraries that provide the basis of the user interface.

Android Runtime offers the Dalvik VM component, which is the Java Virtual Machine on which Android applications are run. Java-based libraries and application frameworks (Application Framework) are in the middle layer.

Some of the services provided by the Application Layer structure serving applications are:

- Activity Manager: Workload stack checks,
- Content Providers: Distributed application data,
- Resource Manager: Resource access to user interface edits,
- Notifications Manager: Permission to manage application notifications,
- View System: Provides services such as application interface settings.

In the Application Layer, which is the top layer, there is software developed using the Android Application Programming Interface (API) to interact with the end user. Permissions such as sending SMS, location notification, reading/writing to external memory requested by applications are authorized through the application framework and communicate with system resources. For example, to perform check-in on Android, at least one of the ACCESS COARSE LOCATION or ACCESS FINE LOCATION permissions must be requested by the applications. If the users approve these permissions, the Location Manager Service, which is one of the middle layer application framework components, provides location access information.

D. MALWARE TERMS AND TARGETS

Many definitions are used to refer to malware, one of which is the words "malicious" and" software", while another is defined as code that causes damage by causing the system to intentionally change its intended task" [25]. In other words, malware is defined as "a general term covering code that interferes with the system, such as viruses, Trojans and spyware" [26]. By other definition, malware is capable of infecting executable code, boot partitions of drives, data, and system files, generating excessive traffic that leads to DoS on the network; are programs that become memory resident when the user executes the file and infect other files that are then executed. In vulnerable operating systems, they can take control of the system and infect other systems on the network. This type of malware program generally affects the performance of the system negatively and causes slowdowns.

Based on the definitions mentioned above; all code that poses a risk to device users, data or devices is classified as malware. Malware aims to corrupt the Android ecosystem and devices with malicious behavior. For this purpose, malware categories are updated every day and new ones are added. Even though malware may differ in behavior, it does show at least one of the targets listed below.

•Creating a security breach by disrupting the holistic behavior of the device,

•Making device control independent from the user,

•Activating device features independent of use with malicious remote access tools,

•Transferring personal data to other areas by abusing consent and intentions,

•To adversely affect other networks and devices with unnecessary commands,

•To defraud device users.

Applications downloaded from various sources can sometimes become the source of malware behavior, even if they do not tend to harm. The applications that cause this are the differences in the ways they work due to the framework changes. In other words, due to Android operating system version differences between devices, malicious software for a device may not pose a risk to the Android device using the new version. Malware, on the other hand, is any version of an Android device or app that could put its users in danger. The malware takes advantage of the device vulnerability and damage the system through unauthorized access by applications.

E. MALWARE TYPES

Malware can be installed on devices due to human or system vulnerabilities, spreading in the system they participate in, keeping the system busy, scaring, stealing, etc. shows typical behavioral traits. The similarities or differences between these behaviors allow classification of malware behaviors as shown in Fig. 6. In addition, classification allows the detection of malicious software according to their potential attack behavior [16], [27]. The types of malwares described below vary in their intended targets and are named according to their behavior.

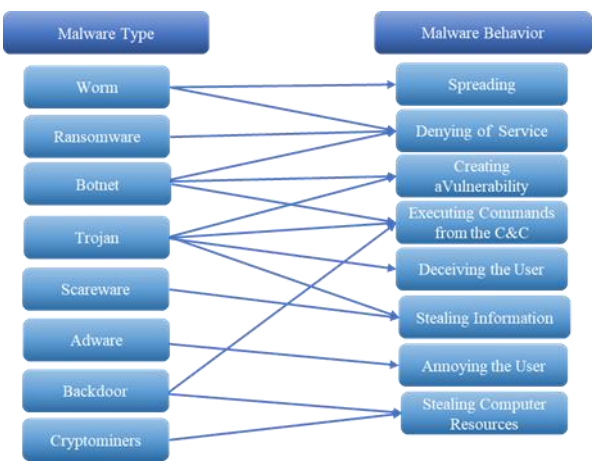

FIGURE 6. **Malware types and behavior.**

Ransomware is an ever-evolving strain of malware designed to encrypt files on a device, rendering files and systems based on them unusable. Malware actors demand a ransom for decryption, threatening to sell or leak personal data or authentication information if the ransom is not paid [27]. These types of software pose an access problem through social engineering methods, such as sending phishing emails or hijacking that system to distribute the malware, which can occur when downloading any file to an unknown recipient. Ransomware has been infecting software installed on many devices, especially mobile devices, in recent years. Ransomware harms Android systems by targeting them [28]. Ransomware types that affect Android systems are generally categorized as lock screen, crypto, and PIN locker. Ransomware, which affects the lock screen, prevents device use with an image that fills the screen completely. Crypto ransomware, on the other hand, is a threat that restricts the user's file access. The PIN locker targets Android devices and changes access codes to lock users. In order not to be affected by the damage caused by this software, it is necessary to use learning-based malware detection systems by keeping the operating systems patched and up to date.

Botnet is a type of attack in which the attacker spreads malware to many devices, breaching their security and taking control of these devices, remotely controlling all the infected devices. It is quite easy to attack the operating systems of Android devices because they are open source. The Android botnet is one of the most important threats to devices. Android botnets are a group of compromised smart devices controlled by remote bot administrators via Command and Control (C&C) servers [29]. The internet permission is the one most needed and requested by all applications, be it good or malicious. In second place is the most READ PRONE STATE permission, which is requested twice by malicious or benign applications. Apart from these permissions, the most requested SMS group permission, which includes SEND SMS, RECEIVE SMS, WRITE SMS and READ SMS, is requested in Botnet specific attacks [30].

Trojan is a malware that acts to provide functions that benefit the user, but instead allows a malware program to be installed on devices. Also, a Trojan has a user dialog that will be constantly activated [31]. Trojans are also known as software that copies and abuses SMS by following the notifications of other applications on the device. It is used to receive messages from the user from other applications and send emails through applications that require two permissions, "Notification Access" and "Internet" [32]. The required permissions for the applications need to be verified. The lack of validation makes the detection and blocking of Trojans problematic. For example, more than half of the apps that require the "READ SMS" permission is classified as malicious, while only some of the apps that require the same permission are benign. Configuration of Android system notifications is app controlled. Android device developers allow the disabling of all or some notifications of the applications to be customized without restriction. However, when configuring these notifications, restrictions may need to be put in place to track activities that may be caused by malware. For example, turning off SMS alerts for banking applications will also prevent uninformed money transfers and suspicious transaction notifications that may be caused by malware actions.

Adware is a type of malware application that targets user privacy and security. This malware harms the user by taking screenshots of them, stealing their data, sending them to a remote server, or forcibly displaying advertisements in the notification bar. The adware can hack the speaker and camera of the smartphone. The main purposes of adware are to collect the data of the websites visited by the user, display customized advertisements on the computer, redirect them to the websites, and collect marketing data. The aggressive adware of adware attacks can create alternative paths on the user's screen, change the default internet browser, internet settings, search engine, and send meaningless notifications. This adware can exploit vulnerabilities through applications downloaded from unofficial sources [33].

Backdoor is a software that provides the basis for the installation structure of other malicious software on the devices that are exposed to the attack. The methods that allow remote access to that computer by a person who bypasses the normal identity verification processes or is aware of this established structure, which cannot be found with ordinary examinations on the computer, is called a backdoor. It is a type of malware that provides a backdoor on victim devices, paving the way for other malware. The backdoor virus hides and runs in the background, making it extremely difficult to detect. This works by taking advantage of the DoS attack. Installing several malware apps containing hidden codes on mobile devices leads to backdoor attacks [34].

A **worm** is a type of software program that produces copies and spreads them over a network. Sharing features such as music, videos, and photos via smart mobile devices or providing users with games, etc. By sending benign-appearing files through applications and directing such files to download, attackers exploit such vulnerabilities to autonomously propagate worms. When users click on the sent link, the worm infects the device and spreads by sending the same message to the people on the infected device and tries to infect other devices. Worm attacks can cause minor or considerable damage to a single user, depending on what the worm is doing. This multiplies proportionally to the number of users who clicked on the link embedded in the message [35].

Scareware is software that encourages the user to buy by scaring them with unrealistic scenarios. It is done to intimidate users by trapping them on phishing web- sites. Scareware is malware that allows the presence of security-threatening elements, such as "against malware programs" or "has malware on the device", to be down- loaded onto the device by presenting legitimate-looking applications. In addition, they harm the system by deceptively presenting many methods, such as deceptive pop-up notifications, fake progress bars, and fake filtering. Also, scareware can cause harm by displaying forged documents that are not on the device or having regulations that are inconsistent with the operating framework [36].

Cryptominers is called attackers infecting computers with viruses using hardware that does not belong to them or deceiving their victims through mining sites. Most of this malware, which have gained popularity recently, are classified as potentially unwanted applications [37].

III. ANDROID MALWARE ANALYSIS TECHNIQUES

Android malware analysis is the process of understanding its code, determining its behavior and functionality, and determining whether it is malicious or not. There are various methods used for this purpose. These methods are categorized as static analysis, dynamic analysis, and hybrid analysis, which is a combination of these two approaches [38], [39].

It is necessary to perform classification processes according to the data obtained by performing malicious software analysis of applications. For this reason, feature extraction processes are performed first for software analysis of Android applications, which naturally contain many features. The process of obtaining the properties requires reverse engineering. For learning-based models, the results obtained by reverse engineering are important. Because the amount of data to be obtained from APKs for analysis purposes is directly proportional to the features to be used in learning models. While Android APKs are input for reverse engineering, the output is usually raw codes, APIs, permissions, purposes extracted from various files. The files obtained from the APK file by reverse engineering are classified in terms of containing static or dynamic features. Extracted features; permissions and intents are expressed as static properties, while network flow, system calls etc. referred to as dynamic properties. The features extracted from the file according to the analysis type shown in Fig. 7 are used to detect malware in learning-based systems and to give effective results in classification [10].

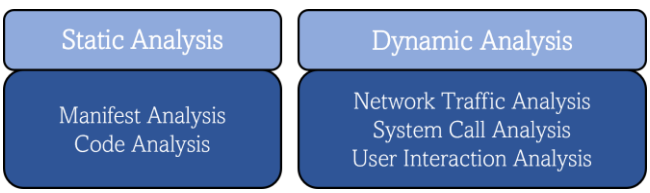

FIGURE 7. Feature extraction methods for static and dynamic analysis types.

Intrusion detection comes automatically because of training static, dynamic, and hybrid analysis features by using learning algorithms. Thus, it facilitates the detection of attacks that cannot be detected manually. Due to the diversity of the extracted information, the features that need to be used to use the categorized data in learning-based attack detection usually need to be transformed into a vectorized representation. In studies in the literature, APKs are processed and analyzed first, and then processing is carried out on the information extracted from the applications [15], [20], [38].

A. STATIC ANAYSIS

Static analysis is a safe method for detecting malware that doesn't require running an application, ensuring that the mobile device remains unaffected by malicious code. This is

just one of the benefits of static analysis. At its simplest, static analysis can gather metadata such as the filename, type, and size of the malware, which can provide insights into its nature, without requiring the code to be viewed. Additionally, MD5 check sums or hashes can be used to compare malware against a database of known malicious software. This approach offers a quick and cost-effective means of identifying harmful features and code in an application before it runs [40], [20]. Below, we describe some of the methods used in the static analysis approach to detect malware, including code analysis, API calls, and permissions.

1) METHODS BASED ON MANIFEST ANALYSIS

The AndroidManifest.xml file is an essential component of the Android app development process. It is a configuration file that contains information on the structure of the application, its components, the rights that are utilized, and the libraries that are necessary. It contains properties that can be used for static analysis. With reverse engineering methods, the permission properties are used by bringing the permission properties into binary property vector form by means of a parser from the Android- Manifest.xml file, which contains the list of all permissions requested by the application [41]. It can be achieved by detecting risky permissions and keywords and examining target applications. It can be determined by a method based on the comparison of the MD5 cryptography hash values of the applications and the suspicion levels assigned to the keywords [42]. These levels of suspicion are the permissions requested by applications as codes containing permissions such as the internet, sending SMS, location access, and writing to a file. These permissions are classified as normal, dangerous, signature, and special permission levels. Normal permissions are low-level user secrets or permissions that affect other applications; dangerous permissions require access to user data and affect the operation of other applications; signature permissions are permissions signed by the same certificate as the permission contained in the application; and special permissions are sensitive permissions that require authorization and require intents that specify permission to the manifest file, requesting user authorization [43].

2) METHODS BASED ON CODE ANALYSIS

Android, a Dalvik registry-based virtual machine, and these applications are developed in Java, compiled into Java bytecode, and then translated into Dalvik bytecode. With the help of byte-code analysis, application behavior analysis, control and data flow analysis, and harmful functions performed by malicious applications are detected. The analysis of these functions can be done by running and debugging line by line, and by searching for specific instruction sets, code analysis can be done. However, the most important factor in code analysis is reverse engineering because of the DLLs, libraries etc. used by the software [54]. Source code conversion is an important part of code analysis. The better the source code conversion is done, the better the code analysis is at figuring out malicious software that has been analyzed and packaged or code obfuscated.

B. DYNAMIC ANALYSIS

Dynamic analysis requires executing the application in an isolated environment to observe application behavior. Unlike static analysis, dynamic analysis allows the natural behavior of malware to be revealed because of the code execution processes. This natural behavior is observed by establishing a virtual environment so that it is not included in other systems; therefore, it requires cost. In this virtual environment, monitoring network activities, monitoring file changes, and determining system calls are performed. Therefore, the functionalities of the actions performed by the analyzed malware require it to be a behavior-based system [20], [45]. The methods used for malware detection in the dynamic analysis approach, including system calls and monitoring of user and kernel level system behavior, are explained below.

1) SYSTEM CALLS ANALYSIS

Android apps interact with the operating system through system calls. It is an analysis technique with data obtained by recording the frequency of clicks, voice interactions, and taps that trigger system calls. When malware makes a system call while it is executing, it intercepts with the hook method to save the function's name and input parameters. While this method is called Function Hooking, the method that pro- vides information about the operation of the malware, in other words, that keeps the sequence of system calls and monitors the results is called Function Call Traces [46]. Also, Android systems is a mobile operating system whose system calls are an interface provided by the Linux kernel. Because Linux kernel system calls are version independent, they are more resistant to avoidance strategies than API calls.

2) USER INTERACTION ANALYSIS

Various activities related to malware, including behavioral analysis of the malware, registry, file system, and network, are constantly monitored. When malware runs as a process, it can perform various operations such as loading images and creating files in the Registry. File system behaviors of malware can be identified by analyzing activities such as reading, creating, and rewriting system files. Additionally, malware behavior can be monitored by working as a fake DNS, HTTP, or FTP server to analyze network traffic [47]. Information related to API-level calls is also monitored at the User-Space Level. If malware software is detected in user mode, the infection can be easily removed by undoing the changes or completely reformatting the system. The kernel, on the other hand, is responsible for managing and allocating system resources as part of the operating system. To monitor malware behavior in the kernel space, the kernel execution traces are analyzed [47], [48]. The kernel space component registers itself with the operating system to track the existence of functions such as processes, registries, and files.

3) NETWORK TRAFFIC ANALYSIS

Malware requests to connect to and/or request commands from the server are often periodically. For example, the most known Plankton software sends a packet every four seconds [49]. Therefore, the average number of packets sent and received per stream, average packet size, and average bytes size application run time are generally low and stable for malware. However, since malware makes requests periodically, it increases network traffic because it is not so periodic in normal internet surfing. Although network traffic analysis, which is frequently used in dynamic analysis technique, is costly in terms of time, it provides high performance in detecting malware apps interact with the operating system through system calls. It is an analysis technique with data obtained by recording the frequency of clicks, voice interactions, and taps that trigger system calls. When malware makes a system call while it is executing, it intercepts with the hook method to save the function's name and input parameters. While this method is called Function Hooking, the method that provides information about the operation of the malware, in other words, that keeps the sequence of system calls and monitors the results is called Function Call Traces [46]. Also, Android systems is a mobile operating system whose system calls are an interface provided by the Linux kernel. Because Linux kernel system calls are version independent, they are more resistant to avoidance strategies than API calls.

C. HYBRID ANALYSIS

Hybrid analysis is an analysis approach that is a combination of static and dynamic analysis methods. Static and dynamic analysis techniques have advantages and dis- advantages compared to each other. For example, while malicious software can often avoid static analysis techniques with code obfuscation methods, it can be detected more easily at execution time with dynamic analysis techniques [11]. The hybrid analysis technique, on the other hand, increases the detection accuracy since it is designed to benefit from both static and dynamic analysis, but takes longer than the run-time dynamic analysis technique because it is computationally intensive. In other words, it is an analysis method that works collaboratively with the static analysis technique to detect the presence of malware behavior with the data obtained during the execution of the application [50].

D. STATIC AND DYNAMIC ANALYSIS FOR DETECTION APPROACHES

Static, dynamic and hybrid analysis are used to detect malware attacks and prevent system infiltration. Attackers use multiple techniques (e.g., bundling, code obfuscation, and encryption) to evade static analysis (signature-based) and dynamic analysis (behavior-based) detection. Static analysis is a passive approach that extracts features from the APK file without running the application. This methodology is cost-effective in terms of resources and time as detection takes place before the application is executed.

Dynamic analysis is an active approach; it describes attacks in the real environment. Permissions, API calls,.dex files, and metadata for transaction codes are examples of static analysis features, whereas dynamic analysis includes network traffic, battery usage, CPU usage, and IP address opcodes. Hybrid analysis includes the features of both static and dynamic analysis methods. Generally, there are two categories of static analysis: Op-code analysis and manifest API call analysis. In Op-Code analysis, malware and benign applications are classified using machine learning algorithms by removing N-Gram Opcodes from the dataset. In manifest and API call analysis, features such as permissions and purposes of the applications are obtained from the manifest file and used [51]. The dynamic detection category consists of two categories: in-the-box analysis and out of the box analysis [19]. In the box analysis, data collection and analysis are treated at the same level as malware. This approach allows capturing data at the operating system level, accessing memory architecture, libraries, APIs, and other methods. But it also makes critical data vulnerable to attacks. In the out-of-the-box analysis approach, it is done by considering the security of the system. A sandbox is created, and analysis behavior pattern is found to fend off the attack.

When the static and dynamic analysis mechanisms are compared, the dynamic analysis mechanism performs better than the static analysis mechanism in detecting attacks using the code hiding technique since the application is analyzed at run time. The static analysis mechanism, on the other hand, is more efficient for detecting previously known attacks. The comparison of the analysis methods, which is about the superiority of the analysis methods over each other, is presented in Fig. 8.

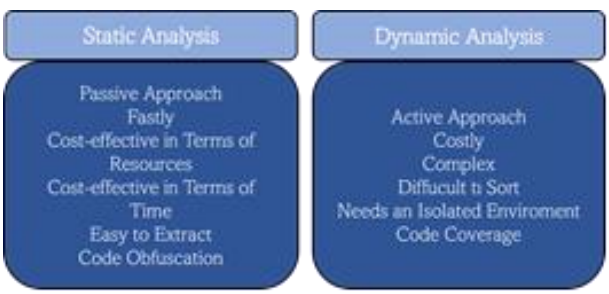

FIGURE 8. Comparison of analysis approaches.

Malware detection systems are generally known to be host-based or network-based [19]. In a host-based system with the detection tool in the system itself, malware activities occurring in the system can be well analyzed. However, it is less able to protect the system against external attacks. They are easy targets for attackers because they are in the system itself. The network-based detection system, on the other hand, can access the entire network and system to which the computer is connected. However, in the network-based detection mechanism, the details of malware activities occurring at the core level of the system are hidden, so they cannot be

analyzed. In addition, it is more costly because it requires more software and hardware resources. A network-based detection system fails to detect previously unknown attacks. The goal of detection system mechanisms is to cut down on the number of false alarms that happen during malware analysis and detection.

E. ANALYSIS DETECTION SYSTEMS

In this survey study, static, dynamic, and hybrid detection approaches used for the detection and prevention of malware in the literature are mentioned. Hybrid analysis tools, which are a combination of static analysis-based, dynamic-based analysis that can detect malware activities at run-time, and a combination of these analysis methods, are offered. These tools, called systems, detect malware at a low alert level by using features extracted from applications according to the analysis method. Some examples of intrusion detection systems in the literature are shown below to give researchers a framework to work with.

TABLE I
ANALYSIS DETECTION SYSTEMS

Analysis Approach	Analysis Method	Source
Taintdroid [64]	Dynamic	Google Play Store
Apptrace [65]	Dynamic	Google Play Store, F-Droid
Artist [66]	Dynamic	Google Play Store
Flowdroid [6]	Static	Google Play Store
Droidsafe [67]	Static	Google Play Store
Amandroid [68]	Static	Google Play Store, AMD Dataset
Cuckoodroid [69]	Hybrid	Google Play Store
Flowintent [70]	Hybrid	Drebin Dataset, Virusshare, Google Play Store, Baidu App Market

TaintDroid is a detection system that monitors how third-party applications downloaded to the Android system access and manipulate users' personal data [52].

AppTrace is to establish the relationship between the behavior of the user and the codes executed by listing the activities and services declared in the Android-Manifest [65].

ARTist is a compiler-based application analysis detection system that does not depend on Android operating system changes and works only at the application layer [54].

FlowDroid, it is a detection system that offers analysis with context, flow, space, and object sensitivity by handling Android-initiated back calls [6].

DroidSafe is an application analytic detection system that uses API calls to analyze sensitive information flow in Android applications [55].

Amandroid, it is a detection system that performs data flow, data dependency and inter-component communication activities analysis for all objects in the Android application component [56].

CuckooDroid is a hybrid detection and classification system, which is a combination of signature detection module as static analysis and abnormal behavior detection module as dynamic analysis [57].

FlowIntent, it is a detection system designed to detect non-functional transmissions at both the software and network level by automatically identifying suspicious transmissions from application visual interfaces. It is included in the hybrid analysis technique in terms of application inspector and traffic analyzer methods [58].

The methods used by the detection systems, whose descriptions are given above, to detect malicious software by running applications, comparing their signatures, or using both methods, and testing resources as datasets are summarized in Table I.

IV. DATASETS

Android malware detection is one of the areas that researchers have focused on in recent years. datasets of applications collected from various sources were created to evaluate technical approaches to malware analysis. datasets created with static or dynamic features obtained from Android applications differ according to the way the good or malicious software they contain is collected or brought together from various sources, and the features they contain. The current datasets from the literature that enable researchers to develop and test detection systems according to their characteristics are explained below.

The CICMalMem2022 dataset is a balanced dataset created for the detection of Spyware, Ransomware and Trojan malware hidden in memory. There are a total of 58,596 records in the dataset, where each malware's families are included. The generated dataset uses the debug mode of memory dumps [59].

CICMalDroid2020 is a new dataset created by collecting more than 17,341 Android samples collected from Contagio security blog, VirusTotal, MalDozer, AMD and other datasets. The analysis results of 13,077 were obtained successfully according to the analysis results of the samples obtained by collecting 17,341 APKs. It consists of static and dynamic data belonging to five different Android application categories. The dataset, which includes samples collected from December 2017 to December 2018, consists of five different categories: Banking Malware, Adware, Mobile Riskware, SMS Malware, and benign. All APK files are collected and executed using CopperDroid. Runtime behaviors are recorded in log files and divided into groups according to static, dynamic, and network traffic behaviors monitored during analysis according to output analysis results. Statically extracted information, intentions; services and permits; frequency numbers for different file types; sensitive API calls and obfuscation events; system calls, binding calls, and composite behaviors; permanently observed behaviors; and PCAP of all network traffic captured during analysis. The CICAndMalDroid2020 dataset contains a total of 17,341 APK files belonging to the five different categories mentioned above. In addition, three CSV files are presented in this dataset. These are for 11,598 APK files belonging to five categories; firstly, it is the csv file that contains the frequencies of binders, system calls and composite behaviors and offers 470 features, and secondly, it is the csv file of system calls that offers 139 features. The

dataset offers 50,621 csv files that contain all the properties vectors of these APK files, such as intents, permissions, sensitive APIs, services [60].

The **CICAndMal2020** dataset is a new dataset containing 200,000 benign and 200,000 malware samples with 14 malware categories and 191 malware families, with a total of 400,000 Android applications. These 14 categories are: Including adware, backdoor, file infector, no categories, Potentially Unwanted Apps (PUA), ransomware, riskware, scareware, trojan, trojan banker, trojan reducer, trojan SMS, trojan spy, and zero-day. Benign applications were collected from the Androzoo dataset, while malware applications were obtained in cooperation with the Canadian Center for Cyber Security (CCCS). VirusTotal was used to label the malware dataset. The classification of malware families is divided into eight categories: media, data collection, hardware, internet connection, actions-events, antivirus, C&C, storage, and settings. This dataset contains static and dynamic analysis data [51]. The features extracted from the AndroidManifest.xml file as static analysis features are Activities, Broad- cast Receivers and Provider, Metadata, Permissions, System Features (such as camera and internet). For dynamic analysis, six categories of features are extracted: Memory, API, Network, Battery, Logcat, Process [62].

The **InvesAndMal2019** [51] dataset, which includes static and dynamic features, includes 426 malware and 5,065 benign labeled samples divided into four categories: Ransomware, Adware, SMS Malware, Scareware. There are family types belonging to each category. For example, it includes family kinds in other categories such as Downgin family, Ewind family belonging to the Adware category. It is presented as the second part of the CICAndMal2017 [63] dataset, which includes more than 10,854 samples created by combining benign samples collected from Google play and malicious samples collected from various sources, published in 2015, 2016, 2017. The InvesAndMal2019 dataset was created by expanding the 80 network flows contained in the InvesAndMal2019 dataset was created by expanding the 80 network flows contained in the CICAndMal2017 dataset, by combining API calls with sequential relations of API calls. This dataset has discrete and continuous features. In the CICAndMal2017 dataset, continuous features are logs, network traffic and API calls, and discrete features are permissions, battery usage, memory dumps and network.

The **AMD** dataset, which contains 24,553 malwares belonging to 71 malware families, consists of adware, backdoor software, ransomware, hacking tools and different types of trojans developed between 2010 and 2016. There is no benign example of AMD. The Web page of the dataset contains detailed information about the characteristics of the malware families in the dataset [64].

AndroZoo is one of the largest datasets of Android apps. It contains 17,188,349 different APKs from applications analyzed as malicious by antivirus programs. In addition, each application is scanned by more than ten antivirus programs and the results are reported [65].

The **DroidCollector** is dataset collected by collecting traffic network data con- taining data from the Drebin project. DroidCollector consists of the traffic data obtained from the control unit, data storage unit, and traffic generation and collection units of network traffic packets on a per-minute basis [66].

The **Drebin** dataset contains 5,560 malware applications collected between August 2010 and October 2012. Drebin dataset also includes samples from the Genome dataset. 5,560 malware applications are divided into 179 families by the publishers of the Drebin dataset [67].

The **GENOM** dataset is an outdated dataset due to the limited resources created by the Android Malware Genome Project and discontinued due to the graduation of the students participating in the project. It is available as a systematically characterized dataset created by collecting 1,260 malware samples, covering most of the existing Android malware families, from its initial release in August 2010 until late October 2011 [33].

Android datasets in the literature are presented in Table II from 2020 to the past. The table includes the publication years of these datasets, the size of the datasets, the family/category information in the dataset, the number of features, the types of feature analysis, and literature examples of the analyzed datasets.

V. DEEP LEARNING MODELS

Since mobile devices have become an indispensable element of daily life, the value of detecting malicious software has become much more important in terms of personal data. Given the prevalence of Android systems on mobile platforms and the threats to this issue, effective malware detection is needed to support the development of reliable detection and classification tools.

The rapid growth of Android malware applications and technologies to evade detection systems is rendering traditional defenses ineffective. Deep learning has become a prominent research area in recent years, taking place in almost every field with its strong feature abstraction ability. The limited capabilities of machine learning are limiting emerging malware detection systems. First, the amount of data increasing day by day requires the system features to be processed and used in the most functional way. Deep Learning algorithms are algorithms that can give the best results with large amounts of data. On the other hand, machine learning algorithms, which can now be described as traditional, yield results with fewer data points [68]. However, this situation is not at a level to meet today's needs because the amount of data is increasing day by day. While deep learning algorithms try to extract features from the data automatically, in machine learning, distinctive features are determined and given to the system.

TABLE II
COMPRESSION OF DATASETS

Dataset	Year	Size	Contents	Category	Feature Type	Analysis Method	Ref
CICMalMem2022	2022	58,596	Family &Category	3 Malware Categories and Benign	Memory Dumps It is balanced, with half of the memory dumps being malicious and the another half being safe.	Dynamic	[59]
CICMalDroid2020	2020	13,077	Category	4 Malware Categories and Benign	11,598 APK and 50,621 Extracted Feature (static information, such as intent actions, permissions,intent consts, files, method tags, sensitive APIs, services, packages, receivers, etc.)	Static & Dynamic	[60]
CICMalDroid2020	2020	400,000	Family &Category	14 Malware Categories and 191 Malware Families	365 Static Features (Activities, API Calls, Intent, etc.) and 141 Dynamic Features (Memory, API, Network Flows, etc.)	Static & Dynamic	[61]
InvesAndMal2019	2019	5,491	Family &Category	14 Malware Categories and 42 Malware Families	8115 Static Feature (Permissions, API Calls, Intent, etc.) and 84 Dynamic Features (Network Flows, Battery States, Log States, Packages, Process Logs, etc.)	Static & Dynamic	[74]
AMD	2017	24,650	Family	71 Malware Families	24,553 APK	Static	[70]
AndroZoo	2017	17,367,766	Family	AndroZoo is a growing collection of Android apps	17,362,964 APK	Static	[65]
DroidCollector	2016	150,099 Benign 196,760 Malware Applications	Family	797 Malware Families	Traffic Data, 683.4 GB Benign,1,062.9 GB Malware	Dynamic	[66]
Drebin	2014	5,560	Family	179 Malware Families	123,453 APK (Permissions, Intents, API Calls and Network Addresses)	Static	[67]
MalGenom	2012	1,260	Family	49 Malware Families	215 Features Extracted from 1,260 APK (Permissions, API Calls, Manifest Permission, Intent)	Static	[33]

Malware applications in Android systems have become a serious threat to users, researchers have developed effective approaches in this area. Although there are studies involving Android malware detection systems in the literature, researchers need an up-to-date and comprehensive survey based on deep learning-based Android malware analysis. In this survey, a literature review focusing on deep learning approaches was conducted, and interests, datasets, and trends related to deep learning-based Android malware analysis were analyzed.

Artificial Intelligence (AI) applications, which have made great progress in the last decade, continue to develop rapidly. Deep learning, on the other hand, is a subset of techniques in artificial intelligence that use neural network to extract patterns from data [69]. In recent years, deep learning has often been applied to malware analysis. Different types of deep learning algorithms such as convolutional neural networks (CNN), recurrent neural networks (RNN), and feed-forward neural networks are implemented in malware analysis for various use cases such as API calls, permissions, intents, HTTP traffic, and network behavior. Deep learning algorithms are divided into three categories: supervised, semi-supervised, and unsupervised according to the form of the learning process [69].

A. SUPERVISED DEEP LEARNING

It is trained using labeled datasets with inputs and expected outputs. Supervised deep learning algorithms are structures that teach to apply it to unforeseen situations by analyzing training data, producing an inference function for mapping unknown new state data. It includes Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory Algorithm (LSTM).

Multi-Layer Perceptron's (MLP) are feed-forward models that have at least three or more layers and multiple perceptron. They are structures that fully connect the neural network by connecting each neuron in the layer to each neuron in the next layer, usually using non-linear activation functions in the hidden layers, the output of each layer being the input of the next layer, and the output layer at the end. The output layer processes the data from the hidden layers and determines the output of the network. MLP is especially preferred in classification and generalization situations [71].

Recurrent Neural Networks (RNN) outperform many related statistical models, such as recurrent neural network

latent Markov models, as the Input data is time-ordered data. Other deep learning methods accept a fixed size vector as input and produce a fixed size vector as output. The feed forward RNN architecture is called "recurrent" [72]. This is because for each element of an array it performs the same task again based on previous outputs.

Long short-term memory (LSTM) is an RNN-based deep learning method. It has a learning architecture that can remember long-term dependencies at random intervals. It is a very successful method, especially in analyzing data that comes in order according to time or events with a certain relationship [73]. LSTM is a modified RNN network proposed to learn long-range dependencies in time-varying models. In general, LSTM is a subject that includes gradient-based learning methods of artificial neural networks and back propagation training. Exploding gradients are (exploding gradients - Exploding gradient problem in machine learning) a quadratic recurrent neural network that solves the problem [74] when large error gradients accumulate and cause huge updates in neural network model weights during training.

Bidirectional Long Short-Term Memory (BiLSTM) model is a bidirectional structure created using LSTM. These networks run inputs from the past to the future and from the future to the past [75]. With this approach, BiLSTM goes backwards and preserves information from future-related situations.

Convolutional Neural Network (CNN) is the most preferred type of super- vised Deep learning in computer image recognition. CNN has multiple layers that process and extract important features from the image. CNN consists of four basic steps. The Pooling Layer uses the pooling process to sample the output of the previous convolutional layer to reduce the effect of small position shift [76].

B. UNSUPERVISED DEEP LEARNING

Unsupervised Deep Learning is trained using datasets that do not have a specific structure. It includes Restricted Boltzmann Machines (RBM), Deep Auto Encoders (DAE) and Deep Belief Networks (DBN).

Restricted Boltzmann Machine (RBM) is a network structure consisting of two layers: input (visible) and hidden. Although neurons in the input layer relate to neurons in the hidden layer, neurons in the same layer are not connected to each other. This structure is hence called the constrained Boltzmann machine. It randomly decides whether to transmit the inputs calculated in each neuron to the next neuron. This algorithm is a network that performs classification, feature detection and prediction calculations by predicting the probability distribution on the inputs [77].

Auto Encoder (AE) feed-forward networks are inefficient for complex multidimensional input definitions. The deep auto encoder (DAE) structure, which is another type of artificial neural network, was created by developing the functional structures of feed-forward networks. DAE provides effective feature extraction for learning by reducing unclassified and unlabeled multidimensional input data. DAE networks consist of encoder, decoder, and hidden layer structures [78].

Deep Belief Networks (DBN) are a special type of deep networks, and they are formed by training the superimposed RBM layers in two stages. The DBN, which is formed by connecting more than one RBM in a row, is learned by training the RBMs that make up its structure, respectively. In this way, a common probability distribution is modeled between the data applied to the input and the hidden layer in between [69].

C. SEMI-SUPERVISED DEEP LEARNING

Semi-supervised learning is a combination of the features of supervised and unsupervised learning algorithms; It includes algorithms in which the number of labeled data is lower in the training data, but there is more unlabeled data, and the algorithms are executed without providing the complete set of rules. In other words, semi-supervised learning is a learning model that represents a middle class between unsupervised learning (without labeled training data) and supervised learning (with labeled training data). Learning accuracy can typically be increased in this model. Algorithms working in this structure are Generative Adversarial Networks (GAN) and Variational AutoEncoder (VAE).

Unlike other machine learning algorithms, the **Generative Adversarial Networks (GAN)** algorithm has the ability to produce as well as learn. The purpose of the algorithm is to train two models simultaneously. These two models consist of the generating model and the discriminant model, which provide data distribution. The discriminant model oversees making sure that the data from the generative model is as accurate as possible [69].

Variational Autoencoders (VAE) are algorithms that generate highly realistic pieces of content of various types, such as deep generative models, images, texts, and sounds, based on well-designed network architectures and intelligent training techniques with large amounts of data. It is an architecture that has both an encoder and a decoder and is trained to make the difference between the encoded data and the original data as small as possible [79].

VI. LITERATURE REVIEW

Recently, there has been significant research on Android malware detection using deep learning techniques. This section aims to summarize the relevant studies and provide a detailed comparison table of their features. Table 3 provides detailed information about each study, including publication year, dataset and algorithms used, accuracy rates, environment, method, and platform.

Kim et al. [80] proposed a malware detection system called MAPAS, which efficiently uses system resources. MAPAS analyzes the behavior of malicious applications using API call graphs and the CNN model to explore the common features of API call graphs of malware. The study compared MAPAS and the Android malware detection approach called MaMaDroid in terms of memory usage, classification speed, and accuracy.

Utku [81] proposed LSTM based detection systems by

analyzing network traffic on mobile applications and comparing them with NB, RF, SVM, MLP, CNN, RNN, and GRU algorithms. The developed LSTM-based deep learning model has been more successful than the other proposed methods with a 95% accuracy rate. In the study, 10 features of 7845 applications obtained from pcap files of 4704 benign and 3141 malicious applications obtained from the DroidCollector project were used.

Fallah and Bidgoly [82] proposed a method based on the LSTM algorithm for malware detection, classification, and new and invisible malware families. In the proposed study, the analysis of network traffic data containing dynamic features was carried out on the CICAndMal2017 dataset. In the study, it was detected with an accuracy rate of 99.96\% immediately after capturing 50 network traffic flows, and an accuracy rate of 80% was obtained in the detection of new malware.

Xing et al., [83] proposed the autoencoder method to analyze a gray-scale image representation of the malware using dimensionality reduction features. In the study, the dataset obtained from categories such as office, video, gaming, finance, photography, and reading, collected from Google Store and VirusShare, with benign and malicious software, was used. When the proposed AE model is compared with machine learning algorithms, the highest accuracy rate of 96% is achieved.

Amer & El-Sappagh [84] proposed a behavioral Android malware odor predictive model in their study. The model is based on reconstructed API calls, permissions, and system call sequences as static and dynamic properties. The proposed model uses the LSTM model to classify snapshots of API and system call sequences. The model is tested against common ransomware attacks on heterogeneous datasets. In the study, in which machine learning algorithms were used for comparison, the highest accuracy rate was obtained at 99.3% with permission classification.

In the proposed study [85], malware detection based on a hybrid classification method was carried out on the CICMalDroid 2020 dataset. In the study, 97.5% accuracy rate was obtained from the use of MLP and RF algorithms applied with hybrid detection techniques of Android malware. In the study, a classification study is presented on the CICAndMal2017 dataset, which uses network traffic features for Android malware detection. By extracting the features from the network traffic with the one- dimensional CNN algorithm, the relationship between the features was determined using the LSTM algorithm. With the model presented in the study, it was stated that the result of binary classification was 99.79% accuracy with CNN-LSTM, 98.9% accuracy with CNN-LSTM in categorical classification, and a 97.29% accuracy rate with CNN-LSTM in family classification [86].

A hybrid DL capable Android malware detection framework is proposed that uses permissions, API calls, and intents to detect malware from Android apps. With the designed approach, CNN and BiLSTM were utilized by using 10-fold cross-validation in classifying malware. The study was carried out on Androzoo and AMD datasets. In the proposed study, hybrid DL models and comparative DL-based algorithms were critically evaluated [87]. The authors proposed a malware classification model to detect Android malware samples, as well as an algorithmic model and an artificial intelligence- based learning solution based on static analysis and feature extraction from source code. The accuracy of the data was found to be approximately 94.4% after 200 rounds of training, and the e-validation set was used to validate the model [88].

Zhu et al. proposed a framework called SEMDroid for the detection of Android malware. To reduce the variety of features, the system using PCA uses the MLP learning algorithm. In the next step, the SVM algorithm was used as a fusion class, and studies were carried out on two different datasets. Accuracy rates were obtained using static properties; an 89.07% accuracy rate was obtained when multi-level static features were used, and a 94.92% accuracy rate was obtained when sensitive data streams were used as static features [89].

In the study, the MobiTive detection system, which is stated to be more efficient and effective compared to existing security engines, is recommended for deep learning-based malware detection on Android mobile devices. In the study, comparative evaluations are presented in five different categories: feature extraction performance, feature selection types of performance, performance accuracy of deep learning algorithms, real-time detection performance accuracy on different devices, and device feature evaluations from MobiTive are presented in [90].

In the study [91], a structure containing a two-layer method is presented to detect malware found in Android applications. While the first layer of the system creates a fully connected neural network model in which static properties are analyzed, the outputs obtained from the first layer are given as input to the second layer, and the analysis of dynamic properties is classified by the proposed CACNN method based on CNN and AutoEncoder algorithms. While the accuracy rate of the first layer of the binary classification system is 95.22%, the classification accuracy performance of the second layer of the system is reported as 99.3%. In addition, in the experimental study in which the proposed system classified malware as categories and families, it was presented that it classified it with an accuracy rate of 98.2% on a category basis and 71.48% on a class basis.

A multi-modal malware detection model called MDNMDroid to explore the potential relationship between permissions by combining two different networks in [92]. Compared to a single network, a multi-mode network has stronger learning ability and can filter more meaningful features to distinguish malware and benign samples. Evaluation results based on the permission dataset collected show that MDNMDroid achieves 93.18% accuracy. In the study, the effect of using convolutional neural networks, especially startup-based and multi-channel architectures, on

network performance for the detection of mobile malware, is mentioned. It has been reported that an accuracy of 92% was obtained by using a multi-channel model on the Koodous dataset [93].

Wenbo et al. have proposed a multi-mode deep learning-based Android malware detection system. In the system where API calls, permissions, hardware components, and intent features of applications are used, deep learning algorithms are modeled in three different ways. In the system in which DNN, CNN, and CNN-GRU deep learning algorithms are used, it is shown that a 98.74% accuracy rate is obtained by using 5,560 malware samples and 16,666 benign samples [94].

In [95] static and dynamic analysis techniques were used to compile API calls, system commands, manifest permissions, and intent attributes of benign and malicious applications from APK files. To obtain the highest performance value with different configuration values of deep learning hyperparameters, the highest accuracy rate of 99% was obtained because of experimental studies. Mu et al. proposed a solution to the malware detection problem in Android systems by extracting the API sequence of the malware with the text processing method in the Cuckoo sandbox. A total of 11000 samples containing 8000 Android malware and 3000 benign applications were used in the study. Among them, Android malware is mostly collected from Virus Share, Google Play Market. Benign apps are usually downloaded from the Android app store. They used the Dalvik analysis method-based Bi-LSTM method, one of the malware static analysis methods, to evaluate the performance of the system. In the study, an accuracy rate of 96.74% was obtained [96].

In [97] the authors carried out malware detection system for Android devices. The classification study is carried out using static analysis features of benign and malicious applications collected from Google Play and Virus Share. The DBN classification framework is presented in the study, in which 331 features, including API calls and permissions, are used. The obtained accuracy success rate has been reported as 94.64%.

In [98] a new preprocessing method is described to solve the long sequence problem that the LSTM model will face to achieve fast training and high accuracy. In the study using the Drebin dataset, an accuracy rate of 95.58% was obtained with the proposed model. In the proposed study, which uses the Deep Learning-based LSTM algorithm for ransomware detection on Android systems, Feature selection was made using eight different machine learning algorithms: Chi Square, CV Attribute Eval, Gain Ratio Attribute, One Attribute Eval, Information Gain Attribute, Significance Attribute Eval, Relief Attribute Eval, and Symmetrical Uncert Attribute Eval. The nineteen features were selected by a simple majority voting process by comparing the results of all feature filtering techniques. In the study, ransomware detection was performed on the CICAndMal2017 dataset. The feature filtration experiment was conducted on WEKA, on a total of 40,000 samples, of which 20,000 were benign and 20,000 were ransomware labeled. It was stated that the accuracy rate obtained in the study was 97% [99].

The Deep Droid framework was proposed in [100] and consists of three phases: the data collection phase, the feature selection phase, and the machine learning phase.120,000 Android apps were evaluated in the study. API calls and permissions were used as inputs in the study. Research was conducted on 100,000 APK files and 20,000 infected APKs collected from Google Play. An accuracy rate of 94% was obtained in the proposed model.

In [101] a factoring machine-based malware detection method is proposed. The model was applied to the DREBIN and AMD dataset and 100% accuracy was obtained using the DREBIN dataset and 99.22% accuracy was obtained using the AMD dataset. It uses an AE-based approach to classify malware. In the study using static analysis methods, API calls and permissions features are used. With the proposed method, it is said that 96.81% of malware can be reached.

The Droid-NNet model is presented as a deep learning-based malware classifier [102]. Experimental studies were conducted using 215 features of malicious and benign applications, using Malgenome-215 datasets containing 3,799 application examples and Drebin-215 datasets containing 15,036 application examples. The proposed framework is evaluated by comparing it with DT, LR, and SVM. To verify the consistency of the model, 10-fold cross-validation was applied to each of the models. The accuracy rate of the proposed framework, Droid-Net, is 98.81%.

A CNN based model for malware detection in the study, API calls and Opcode sequences are used as features [103]. API function calls are used to represent the behavior of the Android application. Bluetooth in the study, user location information, SMS text messages, phone numbers, etc. As such, eleven class packages were selected. These packages also contain many API calls. The attribute size has been reduced by selecting 1500 API calls among the API calls that represent the best in their class according to their entropy. In the study using the Drebin dataset, the accuracy rate of the proposed fusion model was 97.5%.

The MobiDroid framework [104] is a deep learning-based real-time and fast detection system recommended for Android malware detection, is presented. The first part of MobiDroid, which consists of two parts, contains the feature extraction and the related learning model, while the second part transmits the feature vector of the applications downloaded from official and unofficial sources to the detection system. Features that are found in the manifest files of the applications are used as input to the CNN learning model. The accuracy rate of the proposed system was 97.35. Presents a multimodal deep learning framework in which Android malware is detected, and it consists of four parts: raw data extraction, feature extraction, feature creation, and detection processes. The properties extracted in the study are string, opcode, API property, shared library function code, permissions, component, and environmental property. As a result of the

TABLE III
COMPRESSION OF THE RELATED WORKS

Ref	Year	Analysis Method	Learning Model	Features	Dataset Names	Performance Results
[80]	2022	Static	Lightweight CNN	API Calls Graphs	Google Play, AMD Virus Share	%91.27
[81]	2022	Dynamic	LSTM, NB, RF, SVM, MLP, CNN, GRU, RNN	Network Traffic	DroidCollector	%95
[83]	2022	Static	AE	API Calls Permissions	Google Play and Virus Share	%96
[82]	2022	Dynamic	LSTM	Network Traffic	CICAndMal2017	%99.96
[84]	2022	Static & Dynamic	LSTM	API Calls, System Calls Permissions	Heterogeneous dataset	%99.3
[85]	2022	Dynamic	MLP, RF	System calls, Binder Calls Composite Behaviors	CICMalDroid 2020	%97.5
[86]	2021	Dynamic	LSTM, CNN	Network Traffic	CICAndMal2017	%99.79
[87]	2021	Static	CNN-LSTM, BiLSTM- GRU	API Calls, Intents, Permissions	Androzoo, AMD	%99.2
[88]	2021	Static	CNN	Androidmanifest.xml	MalDroid-2020	%94.4
[89]	2021	Static	MLP	Monitoring System Event, Permission-Rate	Android Market, VirusShare	%94.92
[90]	2021	Static	CNN, LSTM, GRU, RNN	Androidmanifest.xml, Properties, API Calls and Opcode Sequences	Google Play Store, VirusShare	%96.78
[91]	2020	Hybrid	CNN- AE	Permissions, Intents, Network Traffic	CICAndMal2017	%99.3
[92]	2020	Static	CNN	Permissions	Google Play and Virus Share	%93.18
[93]	2020	Static	CNN	Permissions	Koodous	%92
[94]	2020	Static	DNN, CNN, GRU	Permissions, API, Intent, Hardware Components	Drebin	%98.74
[95]	2020	Hybrid	DNN	API Call Sequence, System Command, Manifest Permission and ˙Intent	VirusShare	%99.08
[96]	2019	Static	GRU, BGRU, LSTM, Bi-LSTM	API Calls	Google Play Market, Virus Share	%96.74
[97]	2019	Static	DBN	API Calls, Permission	Google Play Market, Virusshare	%94.65
[98]	2019	Static	LSTM	Dalvik Opcode	Drebin	%95.58
[99]	2019	Dynamic	LSTM	APK File	CICAndMal2017	%97
[100]	2019	Static	DBN	API Calls, Permissions	Google Play Market	%94
[70]	2019	Static	FM	Manifest Files and Source Code	Drebin, AMD	%100 on Drebin, %99.22 on AMD
[101]	2019	Static	AE	API Calls, Permissions	Google Play Store, APKpure	%96.81
[102]	2019	Static	Droid-NNET	Permissions, API Calls	Malgenome, Drebin	%98.81
[103]	2019	Static	CNN	API Calls, Opcode	Drebin	%97.5
[104]	2019	Static	DNN	Android Manifest.Xml, API Calls, Opcode Sequences	Google Play, Virusshare, Drebin, Genom, Contagio Mobile Website, Pwnzen Infotech Inc.	%97.35
[71]	2019	Static	DNN	API Calls, Opcode, Permission, Component Feature, Environmental Feature	Virusshare, Malgenome Project,Virustotal	%98
[106]	2019	Static	LSTM	Requested Permissions, Hardware Components, APP Components, Filtered Intents, Used Permissions, Restricted API Calls, Suspicious API Calls, and Network Addresses	Drebin	%99.32
[105]	2018	Static	CNN, LSTM	Android Manifest.xml and dex files	Genom,	%97.4
[107]	2018	Static & Dynamic	LSTM	Battery & Permission, Binder & Permission, Memory, Cpu & Permission, Network & Permission	Malgenome	%99.7 on Dynamic, %97.5 on Static
[108]	2018	Static	CNN	API Calls	Malgenome, Drebin, Playdrone	%96.33 F1 -Score
[110]	2017	Static	DBN	API Method Calls	Genome, Virustotal, Drebin	%95.05 F1 Score

model's performance, the accuracy rate was reached as 98% [71].

The MalResLSTM [105] framework model was proposed to detect malware, and it is presented by a deep residual long-term memory-based system with the Drebin dataset. The

accuracy of the MalResLSTM model was 99.32%. The proposed deep learning based DeepClassifyDroid detection system consists of three steps. These steps are feature extraction, feature placement, and detection. In the first stages, five different feature sets are created using the static analysis method, and in the next stage, CNN-based malware detection is performed. According to the performance result of the proposed study, an accuracy rate of 97.4% was obtained [106].

An LSTM-based system [107] is proposed for Android malware detection based on static and dynamic features. Static analysis used 279 applications in AMD dataset and 279 malicious applications in MalGenome dataset. AndroidManifest.xml file is used for 558 APK using APKTool 2.0.3. Each APK contains detailed permissions as feature vectors with 330 benign and malware class tags. AMD dataset was used for dynamic analysis. Shell scripts with the "adb-monkey" emulator tool were used to analyze APK files and generate random user events in communication with the application interface. The study collected battery, connector, memory, and permissions feature vectors by sending user login mock events to 1330 malicious apps and 408 benign apps in a 5- second time frame. In the study, an accuracy rate of 99.7% in the dynamic analysis method and 97.5% in the static analysis method was obtained with the LSTM model.

Android malware detection framework called MalDozer was proposed in [108] the classification-based study with API calls. MalDozer automatically detects and learns malware and benign with the application's API calls. Automatic feature extraction is offered using raw API call sequences extracted from the DEX assembly. Malgenome and Drebin datasets were used as a mixed dataset containing malware and benign applications. Benign applications are from the PlayDrone dataset [109]. The F1 score rate obtained in the study was 96.33%.

In the study [110], a deep learning-based framework model, the Deep Flow model, is proposed. The performance evaluation of the DBN-based DeepFlow system is presented in the study, which uses a dataset of 8,000 malware applications and 3,000 benign applications collected from VirusShare (which is publicly computer virus repositories on the net) and Genome. In the study, traditional machine learning algorithms and deep learning performances were compared. The F1-Score of the DeepFlow model is reported as 95.05%.

When the literature studies are examined, there is a decrease in the traditional methods in the techniques used for malware detection and shifting to more effective and effective methods. The increasing aggression and complexity of malware reveals that more dynamic systems must be predictive, especially due to zero-day attacks rather than detection. Due to the ever-increasing capabilities of Android devices, it becomes imperative to take comprehensive security measures instead of basic security methods.

In this part of the study, it is presented comparatively in Table III to create a framework for the studies mentioned in the literature summary.

VII. CHALLENGES AND FUTURE DIRECTIONS

The growing number of attacks on Android systems and their focus on stealing valuable information have made Android security a top priority for the cybersecurity community. Malicious software is often used by attackers to gain access to Android devices and target various aspects of these devices. Android users are particularly vulnerable due to the platform's open-source architecture and ease of application development. Based on the insights obtained in the previous sections, we can draw several key lessons and outline some challenges for future research.

Reducing detection times: In order to reduce the impact of malware threats, it is necessary to develop methods that will increase the speed of threat detection and evaluation, and to produce security solutions. Although the use of machine/deep learning models results in some efficient solutions, it will be better to produce some hybrid approaches or parallel detection mechanisms to increase the efficiency of the system. Systems are susceptible to novel forms of attack. Consequently, zero-day attacks must be identified utilizing an anomaly-based detection method and an appropriate model for rapid malware detection.

Supporting other detection systems: Existing antivirus programs that protect Android devices against threats should be supported in the analysis of applications with methods based on dynamic, static and hybrid analysis. These programs should be tools to ensure that applications' actions are only analyzed for compliance with expected security policy, regardless of blacklisting or signature. The development of such software tools is a popular area supported by industrial and academic research.

Development of hybrid analysis detection systems: More work should be done in the future on hybrid analysis techniques, which are a combination of static and dynamic analysis methods. However, dynamic analysis techniques are less preferred than others due to their accessibility and cost. Static analysis can reveal information that cannot be revealed through the dynamic run time process, while dynamic analysis can reveal run time information. This is because it's hard to hide code by taking advantage of both static and dynamic analysis techniques. This makes it important to offer more solutions in this field.

Need for up-to-date datasets: Existing Android-based and publicly available datasets should be expanded for researchers working in the field of malware detection. Datasets created by researchers in my Android malware detection approaches have several limitations. For example, collected malware may be removed by Google play at intervals between two snapshots, or metadata and APKs may change during that time. Therefore, malicious applications collected may not

fully explain the situation when they are removed. This is a limitation of the datasets created by researchers.

Using new and/or deeper models: Android malware detection remains a popular research topic. It is concluded that deep learning-based detection methods in Android systems will still be a trend topic in the near future, and these systems will be faster, stable, robust and agile with the constantly developing new detection technologies.

Using new feature selection strategies for increasing the accuracy rate: Android malware detection utilizes a vast amount of network data, which includes redundant and irrelevant features. This results in extensive training and testing procedures, which cost more resources and produce a low detection rate. Feature selection refers to the process of picking a subset of the most relevant and usable information from an entire dataset, while discarding irrelevant and redundant characteristics, to construct an effective learning system. In order to choose the optimal feature selection methods, such as intelligent agents, evolutionary algorithms, fuzzy techniques, neural networks, rough sets, and swarm optimization techniques, it is crucial to be able to precisely quantify the importance of features to malware detection and the redundancy of features. This decision reduces the system's storage requirements and its processing expenses. The appropriate one is determined by the learning models of the systems and the dataset's structure. Consequently, the selection of these indicators is a complex matter.

Reducing the training time: Almost all learning-based Malware detection systems require a longer training period, which reduces the security system's effectiveness. This problem becomes more important when deep learning models are employed due to the increasing number of layers involved. Either the adoption of new technology or incremental learning models could be the solution. Transfer learning, a model of machine learning, can be used in conjunction with reinforcement learning as an incremental learning strategy. With this strategy, the adaptation of an existing model to a new issue area, particularly with respect to big data platforms, is performed by utilizing prior employment experience to improve the generalization over another. In addition, the majority of developers' favor GPU technology as a hardware-based solution.

We hope this survey study inspires further research in the context of malware detection. In the future, there will be a lot of research to do about interesting features of different malware, how to search for and find out about attacks, how to predict how malware works, how to extract features, and how to make detection more effective.

VIII. CONCLUSION

The secure use of Android devices has become increasingly important due to their significant market share in the digital world. When users cannot interact with their devices effectively, the reliability and usability of smart mobile devices can be inconvenient. Therefore, providing a secure service against malware is a fundamental requirement for the effective use of these devices. Deep learning algorithms, which have emerged with the development of artificial intelligence, can expedite the learning process, and improve the operation of products, technologies, or services. In this survey study, a detailed literature review was conducted to investigate and analyze the application of deep learning approaches in the context of malware detection on Android systems. The study includes comparison tables about datasets, used models and analysis of detection methods, which present many current datasets and articles. Additionally, the study provides an overview of the Android architecture, explains the systems developed for deep learning-based Android malware detection, and high- lights future research trends. Therefore, Android system security and deep learning algorithms are presented as a solution for malware detection in this active research area with many aspects. Furthermore, we also discuss open research issues for android malware detection systems, along with the cross-layer design and different approaches.

REFERENCES

[1] Statcounter:Desktop vs Mobile vs Tablet Market Share Worldwide 2009 to 2022, (2022), https://gs.statcounter.com/platform-market-share/ desktop-mobile-tablet, (accessed May 1, 2023)

[2] S. Kemp: Digital 2022: Global overview report. Datareportal, 2023 (accessed May 1, 2023), https://datareportal.com/reports/ digital-2023-global-overview-report

[3] Statcounter: Mobile Operating System Market Share Worldwide from 2012 to 2022 (2022). https://gs.statcounter.com/os-market-share/mobile/worldwide, (accessed May 1, 2023)

[4] Y. Kanchhal and S. Murugaanandam, "Android Malware an Oversight on Malware Detection Using Machine Learning," 2022 International Conference on Computer Communication and Informatics (ICCCI), pp. 1-5, 2022. doi: 10.1109/ICCCI54379.2022.9741025.

[5] N. Usman, S. Usman, F. Khan, M.A. Jan, A. Sajid, M. Alazab, P. Watters, Intelligent dynamic malware detection using machine learning in ip reputation for forensics data analytics. Future Generation Computer Systems, vol. 118, pp. 124–141, 2021. doi: 10.1016/j.future.2021.01.004.

[6] S. Arzt, S.Rasthofer, C.Fritz, E.Bodden, A. Bartel, J. Klein,Y. Le Traon, D. Octeau, P. McDaniel,: Flowdroid: Precise context, flow, field, object- sensitive and lifecycle-aware taint analysis for android apps. SIGPLAN Not., vol. 49, Art. no. 6, 2014. doi:10.1145/2666356.259429

[7] E. Amer, I. Zelinka, and S. El-Sappagh, "A multi-perspective malware detection approach through behavioral fusion of api call sequence," Computers & Security, vol. 110, p. 102449, 2021. doi: 10.1016/j.cose.2021.102449.

[8] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern Era-A state of the art survey," ACM Compututain Survey., vol. 52, no. 5, 2019. doi:10.1145/3329786.

[9] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, vol. 13, Art. no. 1, 2017. doi:10.1007/s11416-015-0261-z.

[10] E. C. Bayazit, O. K. Sahingoz and B. Dogan, "Malware detection in android systems with traditional machine learning models: a survey,"2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 2020. pp. 1-8. https://10.1109/HORA49412.2020.9152840.

[11] F. Tong, Z. Yan, Z., A hybrid approach of mobile malware detection in Android," Journal of Parallel and Distributed Computing, vol. 103, pp. 22–31, 2017, doi: 10.1016/j.jpdc.2016.10.012

[12] D. Kreuzberger, N. Kühl and S. Hirschl, "Machine Learning Operations (MLOps): Overview, Definition, and Architecture," in *IEEE Access*, vol. 11, pp. 31866-31879, 2023, doi: 10.1109/ACCESS.2023.3262138.

[13] M. N. Al-Andoli, K. S. Sim, S. C. Tan, P. Y. Goh and C. P. Lim, "An Ensemble-Based Parallel Deep Learning Classifier With PSO-BP Optimization for Malware Detection," in *IEEE Access*, vol. 11, pp. 76330-76346, 2023, doi: 10.1109/ACCESS.2023.3296789.

[14] I. Almomani, A. Alkhayer and W. El-Shafai, "An Automated Vision-Based Deep Learning Model for Efficient Detection of Android Malware Attacks," in *IEEE Access*, vol. 10, pp. 2700-2720, 2022, doi: 10.1109/ACCESS.2022.3140341.

[15] N. Sharma and A. L. Sangal, "Machine Learning Approaches for Analysing Static features in Android Malware Detection," *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, Jalandhar, India, 2023, pp. 93-96, doi: 10.1109/ICSCCC58608.2023.10176445.

[16] Y. Liu, C. Tantithamthavorn, L. Li, Y. Liu" Deep learning for android malware defenses: a systematic literature review.", ACM Journal of the ACM (JACM), vol. 37, no. 4, 2022. doi:10.1145/3544968

[17] L. Meijin et al., "A systematic overview of android malware detection," Applied Artificial Intelligence, vol. 36, no. 1, pp.1-33, 2022. doi:10.1080/08839514.2021.2007327

[18] M. Esmail, A. Esmaeilzadeh, Y. Kim, and K. Taghva, "A survey on mobile malware detection methods using machine learning", In: 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0215–0221 ,2022. doi:10.1109/CCWC54503.2022.9720753

[19] P. Bhat and K. Dutta, "A survey on various threats and current state of security in android platform," ACM Comput. Surv., vol. 52, no. 1, pp.1-35, 2019. doi:10.1145/3301285.

[20] E. C. Bayazit, Esra, O. K. Sahingoz, and B. Dogan, "Deep learning-based malware detection for android systems: A comparative analysis," Tehnički vjesnik, vol. 30, no. 3, pp. 787–796, 2023. doi:10.17559/TV-20220907113227.

[21] R. M. G, "On reverse engineering," IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 2, pp. 244–252 ,1985. doi:10.1109/TSMC.1985.6313354.

[22] D. Barrera, J. Clark, D. McCarney, and V. Oorschot, "Understanding and improving app installation security mechanisms through empirical analysis of android", Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM'12, pp. 81–92, 2012. doi:10.1145/2381934.2381949.

[23] H. Kim, T. Cho, G.-J. Ahn, and H. Yi, "Risk assessment of mobile applications based on machine learned malware dataset," Multimedia Tools and Applications, vol. 77, no. 4, pp. 5027-5042, 2018. doi:10.1007/s11042-017-4756-0.

[24] A. Kumar and I. Sharma, "Understanding the Behaviour of Android Ransomware Attacks with Real Smartphones Dataset," 2023 International Conference for Advancement in Technology (ICONAT), pp. 1-5, 2023. doi: 10.1109/ICONAT57137.2023.10080696.

[25] G. McGraw and G. Morrisett, "Attacking malicious code: A report to the infosec research council," IEEE software, vol. 17, no. 5, pp. 33-41, 2000. doi:10.1109/52.877857

[26] A. Vasudevan and R. Yerraballi, "Spike: engineering malware analysis tools using unobtrusive binary-instrumentation," in Proceedings of the 29th Australasian Computer Science Conference Citeseer, vol. 48, pp. 311–320, 2006. doi:10.1145/1151699.1151734.

[27] F. De Gaspari.et al. "Evading behavioral classifiers: a comprehensive analysis on evading ransomware detection techniques," Neural Comput & Applic, vol. 34, pp. 12077–12096, 2022. doi:10.1007/s00521-022-07096-6

[28] M. Masum et al., "ransomware classification and detection with machine learning algorithms,"2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, pp. 0316-0322, 2022. doi: 10.1109/CCWC54503.2022.9720869.

[29] A. M. Almuhaideb and D. Y. Alynanbaawi, "Applications of artificial intelligence to detect android botnets: a survey," IEEE Access, vol. 10, pp. 71737–71748, 2022. doi:10.1109/access.2022.3187094.

[30] S. Anwar et al., "A static approach towards mobile botnet detection," in IEEE, pp. 563–567, 2016. doi:10.1109/ICED.2016.7804708

[31] S. A. Sheikh, M. T. Banday, "Multi-Recipient E-mail Messages: Privacy Issues and Possible Solutions," Advances in Electrical and Computer Engineering, vol.21, no.4, pp.115-126, 2021. doi:10.4316/AECE.2021.04013

[32] H. Abualola, K. Maha, H. Otrok, and A. Mourad, "An android-based trojan spyware to study the notification listener service vulnerability," Procedia Computer Science, vol. 83, pp. 465–471, 2016. doi: 10.1016/j.procs.2016.04.210

[33] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 2012, pp. 95-109. doi: 10.1109/SP.2012.16

[34] C. Li et al., "Backdoor attack on machine learning based Android malware detectors," in IEEE Transactions on Dependable and Secure Computing, vol. 19, no. 5, pp. 3357-3370, 2022. doi:10.1109/TDSC.2021.3094824

[35] M. Omar and M. Dawson, "Research in progress defending Android smartphones from malware attacks," 2013 Third International Conference on Advanced Computing and Communication Technologies (ACCT), Rohtak, India, pp. 288-292, 2013. doi:10.1109/ACCT.2013.69

[36] S. Omeleze and H. S. Venter, "Testing the harmonised digital forensic investigation process model-using an Android mobile phone," 2013 Information Security for South Africa, Johannesburg, South Africa, 2013. pp. 1-8, doi:10.1109/ISSA.2013.6641063.

[37] M. Caprolu, S. Raponi, G. Oligeri, and D. Pietro, "Cryptomining makes noise: Detecting cryptojacking via machine learning," Computer Communications, vol. 171, pp. 126–139, 2021. doi: 10.1016/j.comcom.2021.02.016

[38] M. Gopinath,S.C. Sethuraman, S,"A comprehensive survey on deep learning based malware detection techniques", Computer Science Review, vol. 47, pp. 00529, 2023. doi: 10.1016/j.cosrev.2022.100529.

[39] A. Gombe, et al., "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. "Computers & Security, vol. 73, pp. 235-248, 2018. https://doi.org/10.1016/j.cose.2017.11.006

[40] R. Jin and B. Wang, "Malware Detection for Mobile Devices Using Software-Defined Networking,"2013 Second GENI Research and Educational Experiment Workshop, Salt Lake City, UT, USA, 2013. pp. 81-88, doi:10.1109/GREE.2013.24.

[41] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, Y. Xiang," A survey of android malware detection with deep neural models", ACM Computing Surveys (CSUR), vol.53, no. 6, pp. 1-36, 2020. doi:10.1145/3417978.

[42] S.H. Seo, A. Gupta, A. M. Sallam, E. Bertino, and K. Yim, "Detecting mobile malware threats to homeland security through static analysis," Journal of Network and Computer Applications, vol. 38, pp. 43–53, 2014. doi: 10.1016/j.jnca.2013.05.008

[43] B. Sanz et.al.,"MAMA: manifest analysis for malware detection in Android"Cybernetics and Systems,vol.44, no.6,pp. 469-488, 2013. doi:10. 1080/01969722.2013.803889

[44] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in IEEE, pp. 329–333, 2010. doi:10.1109/CIS.2010.77.

[45] S. Yerima, Alzaylaee, M. & Sezer, S. Machine learning-based dynamic analysis of Android apps with improved code coverage. EURASIP J. on Info. Security,vol 2019, no.1, pp.1-24, 2019. doi:10.1186/s13635-019-0087-1

[46] G. Suarez-Tangil, G., J.E. Tapiador, P. Peris-Lopez, A. Ribagorda," Evolution, detection and analysis of malware for smart devices", IEEE Communications Surveys & Tutorials,vol. 16, no. 2, pp. 961-987, 2013. doi:10.1109/SURV.2013. 101613.00077

[47] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching", pp. 1–4.2010. doi:10.1145/1852666.1852716

[48] J. Rhee, R. Riley, D. Xu, and X. Jiang, "Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory", pp. 178–197, 2010. doi:10.1007/978-3-642-15512-3_10

[49] A. Arora, S. Garg, S.K. Peddoju," Malware detection using network traffic analysis in android based mobile devices", In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 66–71 ,2014. doi:10.1109/NGMAST.2014.57

[50] S. Zhao, X. Li, G. Xu, L. Zhang, and Z. Feng, "Attack tree based android malware detection with hybrid analysis," In: IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 380–387. doi:10.1109/TrustCom.2014.49

[51] L. Taheri, Abdul, and Lashkari, Arash Habibi, "Extensible android malware detection and family classification using network-flows and API-Calls", In: 2019 International Carnahan Conference on Security Technology (ICCST), pp. 1–8, 2019. doi:10.1109/CCST.2019.8888430

[52] W. Enck and P. Gilbert, "gon chun," B., Cox, LP, Jung, J., McDaniel, P., Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones, ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, pp. 393–407, 2010. doi:10.1145/2619091

[53] L. Qiu, Z. Zhang, Z. Shen, and G. Sun, "AppTrace: Dynamic trace on android devices," in IEEE, pp. 7145–7150, 2015. doi:10.1109/ICC.2015.7249466

[54] M. Backes,S. Bugiel, O. Schranz, P. Von Styp-Rekowsky, S. Weisgerber, "Artist: The android runtime instrumentation and security toolkit", In: 2017 IEEE European Symposium on Security and Privacy (EuroSP), pp. 481–495, 2017. doi:10.1109/EuroSP.2017.43

[55] M. I. Gordon, D. Kim, J. H. Perkins, L. Gilham, N. Nguyen, and M. C. Rinard, "Information flow analysis of android applications in droidsafe", In: NDSS, vol. 15, no. 201, pp. 110, doi:10.14722/ndss.2015.23089

[56] F. Wei, S. Roy, and X. Ou, "Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps," ACM Transactions on Privacy and Security (TOPS), vol. 21, no. 3, 2018. doi:10.1145/3183575

[57] X. Wang, Y. Yang, Y. Zeng, C. Tang, J. Shi, and K. Xu, "A novel hybrid mobile malware detection system integrating anomaly detection with misuse detection", In: Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services, pp. 15–22, 2015. doi:10.1145/2802130.2802132

[58] H. Fu et al., "Towards automatic detection of nonfunctional sensitive transmissions in mobile applications," IEEE Transactions on Mobile Computing, vol. 20, no. 10, pp. 3066-3080, 2021. doi:10.1109/TMC.2020.2992253

[59] T. Carrier, P. Victor, A. Tekeoglu, A.H. Lashkari, "Detecting Obfuscated Malware using Memory Feature Engineering", In: Proceedings of the 8th International Conference on Information Systems Security and Privacy, ICISSP 2022, pp. 177-188, 2022. doi:10.5220/ 0010908200003120.

[60] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," J. Netw. Syst. Manage., vol. 30, no. 1, pp. 1-34, 2022. doi:10.1007/s10922-021-09634-4

[61] A. Rahali, Lashkari, Arash Habibi, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "DIDroid: Android malware classification and characterization using deep image learning", In 10th International Conference on Communication and Network Security, pp. 70–82 ,2020. doi:10.1145/3442520.3442522

[62] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon, and F. Massicotte, "EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics,"2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), pp. 1-12, 2021. doi:10.1109/RDAAPS48126.2021.9452002.

[63] H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification,"2018 International Carnahan Conference on Security Technology (ICCST), pp. 1-7, 2018. doi:10.1109/CCST.2018.8585560.

[64] F. Wei, Y. Li, S. Roy, S., X. Ou, W. Zhou, "Deep ground truth analysis of current Android malware" In Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Springer International Publishing, pp. 252-276, 2017. https:// doi.org/10.1007/978-3-319-60876-1 12

[65] K. Allix, Bissyandé, Tegawendé F, J. Klein, and L. Traon, "AndroZoo: Collecting millions of android apps for the research community", In: 13th International Conference on Mining Software Repositories, pp. 468–471, 2016. doi:10.1145/2901739.2903508

[66] D. Cao et al., "DroidCollector: A high performance framework for high 78quality Android traffic collection", 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 1753-1758, 2016. doi:10.1109/TrustCom.2016.0269.

[67] D. ArpDaniel, et al., "Drebin: Effective and explainable detection of android malware in your pocket.", Ndss, vol. 14., pp. 23–26, 2014. doi:10.14722/ndss.2014.23247

[68] E. C. Bayazit, E.C., O. K. Sahingoz, B. Dogan, B." Neural network-based Android malware detection with different ip coding methods, In: 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1–6, 2021. doi:10.1109/HORA52670.2021. 9461302

[69] S.M. Bohte, "The evidence for neural information processing with precise spike-times: A survey", Natural Computing, vol. 3, pp. 195–206 ,2004, doi:10.1023/B: NACO.0000027755.02868.60

[70] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang, and H. Kinawi, "Android malware detection based on factorization machine," IEEE Access, vol. 7, pp. 184008–184019, 2019. doi:10.1109/ACCESS.2019.2958927

[71] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," IEEE Transactions on Information Forensics and Security, vol. 14, no. 3, 2019. doi:10.1109/TIFS.2018.2866319

[72] J.Seungho, and J. Moon. "Malware-detection method with a convolutional recurrent neural network using opcode sequences", Information Sciences, vol. 535, pp. 1-15, 2020. doi: 10.1016/j.ins.2020.05.026

[73] F.A. Gers, F.A., J.A. chmidhuber, F.A. Cummins "Learning to forget: Continual prediction with lstm", Neural Computation, vol. 12, no. 10, pp 2451–2471, 2000. https://doi. org/10.1162/089976600300015015

[74] A. N. Jahromi, S. Hashemi, A. Dehghantanha, Parizi, Reza M, and K.-K. R. Choo, "An enhanced stacked LSTM method with no random initialization for malware threat hunting in safety and time-critical systems," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 4, no. 5, 2020. doi:10.1109/TETCI.2019.2910243

[75] G. Shen, Z. Chen, H. Wang, H. Chen, H., S.Wang, "Feature fusion-based malicious code detection with dual attention mechanism and BiLSTM.", Computers & Security, vol. 119, pp. 102761, 2022. doi: 10.1016/j.cose.2022.102761

[76] F. Martinelli, F. Marulli, F. Mercaldo, F. "Evaluating convolutional neural network for effective mobile malware detection.", Procedia computer science, vol. 112, pp. 2372-2381, 2017. doi: 10.1016/j.procs.2017.08.216

[77] A. Fischer, C. Igel, C. "Training restricted Boltzmann machines: An introduction", Pattern Recognition, vol.47, no. 1, pp 25-39,2014. doi: 10.1016/j.patcog.2013. 05.025

[78] Y. LeCun, Y., Bengio, G. Hinton, G, "Deep learning", Nature, vol. 521, no7553, pp 436-444, 2015. doi:10.1038/nature14539

[79] V. Engelen, E. Jesper and H. H. Hoos. "A survey on semi-supervised learning", Machine learning, vol.109, no. 2, pp. 373-440,2020. doi:10.1007/s10994-019-05855-6

[80] J. Kim, Y. Ban, E. Ko, H. Cho, J.H. Yi, "MAPAS: a practical deep learning-based android malware detection system", International Journal of Information Security, vol. 21, no. 4, pp 725-738, 2022. doi:10.1007/s10207-022-00579-6

[81] U.T. Anıl "Ağ trafiği analizi ile derin öğrenme tabanlı Android kötücül yazılım tespiti", Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, vol.37, pp. 4, pp. 1823-1838,2022. doi:10.17341/gazimmfd.937374

[82] S. Fallah, A.J. Bidgoly, "Android malware detection using network traffic based on sequential deep learning models.", Software: Practice and Experience, vol.52, no.9, pp 1987-2004,2022. doi:10.1002/spe.3112.

[83] X. Xing, X., Jin, H. Elahi, H. Jiang, G. Wang, "A malware detection approach using autoencoder in deep learning", IEEE Access, vol. 10, pp. 25696-25706 ,2022. doi:10.1109/ACCESS.2022.3155695

[84] E. Amer and S. El-Sappagh, "Robust deep learning early alarm prediction model based on the behavioural smell for android malware," Computers & Security, vol. 116, p. 102670, 2022. doi: 10.1016/j.cose.2022.102670

[85] F. Ahmed, et al. "ShielDroid: A hybrid approach integrating machine and deep learning for Android malware detection."2022 International

Conference on Decision Aid Sciences and Applications (DASA). IEEE, pp. 911–916 ,2022. doi:10.1109/ DASA54658.2022.9764984
[86] M. Gohari, S. Hashemi, and L. Abdi, "Android malware detection and classification based on network traffic using deep learning", pp. 71–77, 2021. doi:10.1109/ICWR51868.2021.9443025
[87] I. U. Haq, T. A. Khan, and A. Akhunzada, "A dynamic robust DL-Based model for android malware detection," IEEE Access, vol. 9, pp. 74510–74521, 2021. doi:10.1109/ACCESS.2021.3079370
[88] B. H. Tang et al., "Android malware detection based on deep learning techniques", pp. 481–486, 2021. doi:10.1109/PRAI53619.2021.9551073
[89] H. Zhu, Y. Li, R., Li, J Li, J., Z. You, H Song, "Sedmdroid: An enhanced stacking ensemble framework for android malware detection", IEEE Transactions on Network Science and Engineering, vol.8, no. 2, pp. 984–994 ,2021. doi:10. 1109/TNSE.2020.2996379
[90] R. Feng et al. "A performance-sensitive malware detection system using deep learning on mobile devices, "IEEE Transactions on Information Forensics and Security, vol. 16, pp. 1563-1578, 2021. doi:10.1109/TIFS.2020.3025436
[91] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A two-layer deep learning method for android malware detection using network traffic," IEEE Access, vol. 8, pp. 125786–125796, 2020. doi:10.1109/ACCESS.2020.3008081
[92] W. Gu, "A multimodal deep network model for android malware detection using permission,", In: 2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI), pp. 63–67, 2021. doi:10.1109/ICETCI53161.2021.9563414
[93] F. Bourebaa, M. Benmohammed," Android Malware Detection using Convolutional Deep Neural Networks", In 2020 International Conference on Advanced Aspects of Software Engineering (ICAASE), pp. 1-7, 2020. doi:10. 1109/ICAASE51408.2020.9380104
[94] F. Wenbo, Z. Linlin, W. Chenyue, H. Yingjie, Y. Yuaner and Z. Kai, "AMC-MDL: A novel approach of android malware classification using multimodel deep learning,"2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, pp. 251-256, 2020. doi:10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00052.
[95] R. B. Hadiprakoso, I. K. S. Buana and Y. R. Pramadi, "Android malware detection using hybrid-based analysis & deep neural network", 3rd International Conference on Information and Communications Technology (ICOIACT), pp. 252-256, 2020. doi: 10.1109/ICOIACT50329.2020.9332066.
[96] T. Mu, H. Chen, J. Du and A. Xu, "An Android malware detection method using deep learning based on apı calls,"2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), pp. 2001-2004 ,2019. doi:10.1109/IMCEC46724.2019.8983860.
[97] S. HR, "Static analysis of Android malware detection using deep learning", International Conference on Intelligent Computing and Control Systems (ICCS), pp. 841-845, 2019. doi: 10.1109/ICCS45141.2019.9065765.
[98] Y. M. Chen, C. H. Hsu and K. C. Kuo Chung, "A novel preprocessing method for solving long sequence problem in Android malware
[99] detection,"2019 Twelfth International Conference on Ubi-Media Computing (Ubi-Media), pp. 12-17, 2019. doi:10.1109/Ubi-Media.2019.00012
[100] I. Bibi, A. Akhunzada, J. Malik, G. Ahmed, and M. Raza, "An effective android ransomware detection through multi-factor feature filtration and recurrent neural network,"2019 UK/ China Emerging Technologies (UCET), pp. 1-4, 2019. doi: 10.1109/UCET.2019.8881884
[101] A. Mahindru and A. L. Sangal, "DeepDroid: Feature Selection approach to detect Android malware using Deep Learning,"2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), pp. 16-19, 2019. doi: 10.1109/ICSESS47205.2019.9040821.
[102] A. Naway, Y. Li, "Android malware detection using autoencoder", International Journal of Computer Engineering and Applications, vol. 12, no. 12, 2018. doi:10.48550/arXiv.1901.07315
[103] M. Masum and H. Shahriar, "Droid-NNet: Deep learning neural network for android malware detection,"2019 IEEE International Conference on Big Data (Big Data), pp. 5789-5793, 2019. doi: 10.1109/BigData47090.2019.9006053.
[104] Y. Ding, J. Hu, W. Xu and X. Zhang, "A deep feature fusion method for Android malware detection,"2019 International Conference on Machine Learning and Cybernetics (ICMLC), pp. 1-6, 2019. doi: 10.1109/ICMLC48188.2019.8949298.
[105] R. Feng et al., "MobiDroid: A performance-sensitive malware detection system on mobile platform,"2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 61-70, 2019. doi:10.1109/ICECCS.2019.00014
[106] Y. Zhang, Y. Yang, and X. Wang, "A novel android malware detection approach based on convolutional neural network", In: 2nd International Conference on Cryptography, Security and Privacy, pp. 144–149, 2018. doi:10.1145/3199478.3199492
[107] A. Alotaibi, "Identifying malicious software using deep residual long-short term memory", in IEEE Access, vol. 7, pp. 163128-163137, 2019. doi:10.1109/ACCESS.2019.2951751
[108] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning" Digital Investigation, vol. 24, pp. 48–59, 2018. doi: 10.1016/j.diin.2018.01.007.
[109] N. Viennot, E. Garcia, J. Nieh," Playdrone: A measurement study of google play", ACM International Conference on Measurement and Modeling of Computer Systems, pp. 221–233, 2014. doi:10.1145/2591971. 2592003.
[110] R. Vinayakumar, K. Soman, P. Poornachandran, S. Sachin Kumar," Detecting android malware using long short-term memory (lstm)", Journal of Intelligent & Fuzzy Systems, vol. 34, no. 3, pp. 1277–1288, 2018. doi:10.3233/JIFS-169424.

FIRST A. AUTHOR received the B.E. degrees in Computer and Control Education department from Marmara University, Istanbul, Turkey and Computer Engineering from the Sakarya University, Sakarya, Turkey, in 2010 and 2019, respectively, and M.E. degrees in Computer and Control Education department from Marmara University, Istanbul, Turkey, in 2013, and the Ph.D. degree in Computer Engineering from the Marmara University in 2023. She has been working as Assistant Professor Fatih Sultan Mehmet Vakif University. Her research interests include computer networks and security, machine/deep learning, and python programming.

SECOND B. AUTHOR received the B.Sc. degree from the Computer Engineering Department, Bogazici University, in 1993, and the M.S. and Ph.D. degrees from the Computer Engineering Department, Istanbul Technical University, in 1998 and 2006, respectively. He is currently working as Professor with the Computer Engineering Department, Biruni University/Istanbul. His research interests include artificial intelligence, machine/deep learning, data science, and UAV networking.

THIRD C. AUTHOR, received the B.Sc. degree from the Marmara University, Faculty of Technical Education, Department of Electronics and Computer Education, Turkey in 1999 and the M.S. and Ph.D. degrees from Marmara University, in 2001 and 2006, respectively. She is currently working as Associate Professor with the Computer Engineering Department, Marmara/Istanbul. Her research interests include data mining, artificial intelligence, data science.