A Review and Case Study on Android Malware: Threat Model, Attacks, Techniques and Tools

Charu Negi[1], Preeti Mishra[2,*], Pooja Chaudhary[2] and Harsh Vardhan[2]

[1]*Graphic Era Hill University, Dehradun, India*
[2]*Graphic Era Deemed to be University, Dehradun, India*
E-mail: chandola.charu@gmail.com; scholar.preeti@gmail.com; poojachaudharypcm@gmail.com, harshsam80@gmail.com
[*]*Corresponding Author*

Received 10 September 2020; Accepted 31 October 2020;
Publication 11 March 2021

Abstract

As android devices have increased in number in the past few years, the android operating system has started dominating the smartphone market. The vast spread of android across all the devices has made security an important issue as the android users continue to grow exponentially. The security of the android platform has become the need of the hour because of an increase in the number of malicious apps and thus several studies have emerged to present the detection approaches. In this paper, the android components have been reviewed to propose a threat model that illustrates the possible threats that are present in the android. The researchers also present the attack taxonomy to illustrate the possible attacks at various layers of the android architecture. Experiments demonstrating the feature extraction and classification using machine earning algorithms have also been performed.

Keywords: Android, android architecture, attack taxonomy, malware detection, machine learning, malware.

Journal of Cyber Security and Mobility, Vol. 10_1, 231–260.
doi: 10.13052/jcsm2245-1439.1018

1 Introduction

Android is an open-source mobile operating system and is based on the Linux kernel. Android has gained significance in recent years in the market and the number of Android users has increased significantly [1]. At this point, it becomes very necessary for the developers to take care of the security and all the possible threats to the private data of the android user. Android being an open-source operating system has gained the attention of many hackers and attackers as well. Android applications are the software for android operating systems which can be downloaded from the Google play store and by using third party applications also. It is important to note that as of March 2020, the number of malware samples per month is almost half a million [2]. This growth is directly proportional to the increase in the number of Android users.

It is however a disturbing fact that the number of users unwary of the possible malware apps is more than aware users. The malware writers make use of this situation to install their malicious programs on the devices of such users [2]. As the number of malware apps begins to explode, it becomes important to understand how malware apps function. Android is an open-source operating system and thus provides numerous opportunities to developers and malware writers alike. The developers make use of this opportunity to improve their experience by downloading e.g. a different operating system than the one provided or making some code changes to suit their needs. The attackers modify the code to exploit the vulnerabilities [3] to gain control for malicious intentions.

Android is vulnerable to attacks in any of its architecture layers such as the kernel layer, application layer, etc. Existing code defects or issues provide attackers an attack surface to launch their attacks. Research in the direction of malware analysis includes studying the different components of Android OS and an Android application [4]. The analysis encompasses the usage of features obtained statically or dynamically and then using state of art technologies, to name a few, machine learning algorithms, or deep learning [5]. These approaches help identify the presence of malware and the family it belongs to [6]. The different approaches studied included advantages over the other and also limitations faced.

Malware Analysis is important in Android as the usage of mobile devices increases by the day. It is important to identify any threats emerging for these devices and stop them from doing any damage to the device or user of the same [7]. This article provides an in-depth explanation of the android architecture and threat model of android to get a better understanding of how

android apps may be affected. Understanding android architecture becomes important to detect malware apps. The threat model of android becomes important for understanding the vulnerabilities of android and how a breach can happen due to various reasons. Case studies have been included in this article to provide a clear understanding of the features and classification [8]. Android implementation being an open market model, can lead to various breaches and put sensitive information of users at stake.

The major contribution in this paper is

- Propose a threat Model that helps understand the Android vulnerabilities
- Present an attack taxonomy for Android for each of the layers
- Describe the various detection approaches and tools to provide a deep understanding that help analysis using these techniques
- Demonstrate the feature extraction process and machine learning implementation for classification

The rest of the paper is organized as follows – in Section 2 the background about android by explaining its architecture and then the layout of an android apk has been discussed. Section 3 presents a threat model followed by attack taxonomy in Section 4. Section 5 includes details for the detection approaches, its tools, and case studies. Section 6 details the related work for Android malware analysis and techniques. In the end, Section 7 presents the future directions and 8 concludes the paper.

2 Android – The Background

Before beginning the analysis of Android apps for identifying whether it is a malware or not, reader should be able to understand the Android architecture and an Android app's structure. This section includes details about the architecture of Android presenting all the different components of the stack-based Android operating system. An Android app consists of different types of artifacts such as files and folders which contribute to the working of the app. These files present a major source of information to the malware analyst as several important features can be used for malware analysis.

2.1 Android Architecture

It is important to understand the basic architecture of Android to understand its different components and how these are organized. Android architecture

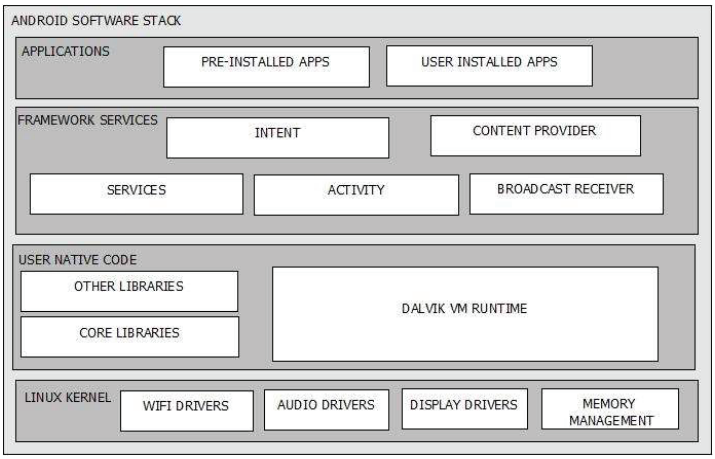

Figure 1 Android architecture.

[9][10] mainly consists of 5 layers as shown in Figure 1. The following text explains the different layers and their components.

- The Android Applications – Applications in android are of two types: Pre-Installed- These include original equipment manufacturer (OEMs), calculators, etc. The package of these apps resides at /system/app/directory. Second is user-installed-these are installed by the user from app markets like Playstore, etc. These apps and updates to pre-installed apps reside at /data/app directory. Android uses a special platform key for pre-installed apps packages. Apps signed with this key are special as they have system user privileges. For both pre-installed and user-installed apps, android uses a signature to prevent unauthorized app updates.
- Android Framework – The Android framework consists of different packages and classes that the developer can implement for performing common functions. It is like an interface between the app and the runtime environment.
- Application Components – There exists four main components discussed as follows. *Activities* – It enables the user of the system to be able to take action on the smart device. It is in the form of an application component or UI and is available for the user to view. *Broadcast Receivers* – It handles the communication established between the android operating system and android apps. These are useful in case of implicit intents and apps can receive intent corresponding to a specific

task. *Services* – It handles background processing that is associated with applications. E.g. when someone downloads something from a website. *Content Providers* – It handles the data and database management of an android application. Some additional components include Intents. These are message objects that are sent from one app component to the other. These contain information such as the action to be performed, the component to which the intent is being targeted, and some additional flags. All common actions involve intents to be passed around the system.

- Dalvik Virtual Machine – The Dalvik VM is a virtual machine for Android devices. It is register-based, as opposed to JVM which is stack-based. Its syntax looks like JAVA and also it works the same as JVM but it cannot be used because of its large memory consumption which is not possible in small devices like Smartphones, Tablets, etc. The overall development process is that first a developer writes code in what syntactically looks like JAVA code. This source code compiles into a .class file. The resultant class file is then converted into Dalvik byte code. All class files in the source code together form the classes.dex file which is the executable file for the DVM. The byte code is loaded and then interpreted by Dalvik VM. The zygote is the first process that starts as soon as the device boots up. This process has the responsibility of loading and starting all the other services and libraries that are to be used by the Android. Thus, the Zygote is a loader for the different apps and works by creating copies of itself. This also creates some interesting areas of attacks. Zygote also starts System server processes. If it is killed, then the system comes into a reboot state.
- Native Code- Android system consists of several Native Code based files. It consists of Libraries that may be vendor-specific and non-vendor specific. It also contains Core system services that setup the underlying OS environment and native android component. These libraries are susceptible to memory-based attacks because of code-based vulnerabilities in the Native code files.

2.2 Android Application Structure

It is important to understand the Android package structure before beginning with the reverse engineering of the app. An Android app is packed into an APK file consisting of several files and folders. In particular, the *Android manifest.xml* file stores the app's metadata such as package name, required

permission, components like activities, services, broadcast services, content providers, etc. The folder *res* stores icons, images, string/numeric/color, consonants, UI layouts, menus, animation compiled into binary. The folder *assets* contain the application's assets which can be retrieved by AssetManager. The executable files *classes.dex* stores the Dalvik byte code files that are understandable to Dalvik VM. The *Meta-INF* file contains the information related to the signature of the app developer certificate. These are explained as below
AndroidManifest.xml: Every Android app contains a Manifest XML which contains the following:

- unique package name and version information.
- Activities, Broadcast Receivers, Services, and Instrumental definitions.
- Permission requested
- Information on external libraries packaged and used by the application.
- Additional supporting directives such as UID preferred installation locations and UI.

META-INF: It is a directory that contains:

- MANIFEST.MF: the manifest file
- CERT.RSA: the certificate of the application
- CERT.SF: the list of resources and digest of corresponding lines in MANIFEST.MF file

3 Android Threat Model and Attack Surfaces

Android is an open-source Operating System for mobile applications that can easily be attacked by hackers. The adoption of an open market model has made Android vulnerable to attacks from outside. Users have easy access to alternative app markets other than Google's play store from where apps that are not available on the play store, can be downloaded [11]. There is no implementation of isolation mechanisms for apps downloaded from third parties. These apps can easily abuse the granted permissions and affect other apps [12]. The Android Debug Bridge can help users install an app without having to use any app markets. Using a PC, any app can be installed on the android device. The attacks on Android are very different from attacks on operating systems, computers, or servers. The reason is that mobile phones are portable which leads to the possibility that they can easily be lost or stolen.

They are portable and thus can be connected to untrusted networks or Wifi. Mobile phones are an integral part of users lives and also carry their

sensitive and private data compared to any other computer, server, or laptop. Some of the common mistakes committed by the users of the device are discussed in the following section that becomes a threat to the device and can make the device vulnerable. Granting of Permission: One of Android's main security measures is asking for Permissions at the time of installation. It can be cumbersome for the user to understand each permission the application asks for. The users tend to give permissions without understanding them. This gives malware applications, easy access to the android device.

Rooting of the device: To gain more control of their devices users tend to root their devices to achieve customization not essentially provided by the Android system. While rooting the device provides users with more permissions to use their device, it can also lead to personal data being exposed to infiltration [13]. *Network Level Threat*: Connection over insecure network or fake access point of a network can be the two possible network-level threats. Connection over an unsecured Wifi network means that connection is allowing the movement of data without any kind of encryption i.e. in the form of plain text which is very dangerous. Any hacker present within a certain range can eavesdrop and collect the information with the use of proper tools and knowledge. *Fake access points*: Hackers or attackers can create fake access points. One may misunderstand these fake access points like the one provided by some standard organization but it is not the case. They are made by hackers to access the private information of the user.

Phishing: Phishing is a technique in which the user of the application gets fooled by the fake SMS or emails. The user responds to these emails and provides all the personal information which leads to easy access of the user account to the attacker. Another form of advanced phishing attack allows the hackers or attackers to act as network operators and send Open Mobile Alliance Client Provisioning messages to the user. *Crypto Jacking attack*: Crypto-jacking is a technique by which the hacker uses the victim's device without his/her consent for mining the cryptocurrency. The battery performance of the android device suddenly deteriorates and the system shows that the android device is less on space. Sometimes, this kind of attack also leads to overheating of the components of the android device. *Broken Cryptography*: This type of threat comes into the picture when the developer has either used a very weak algorithm in the security of the app or has used a strong algorithm but has implemented it in an insecure manner.

Data Leakage: Data leakage is also known as unintended data leakage. It takes place when a piece of certain important information that is taken as

an input by the device is placed in an insecure place in the device and the malicious apps access that information. It becomes easy for the attacker to access this information by writing a small piece of code, the attacker can easily track the location and then access the private data of the victim. There are many different possible scenarios of data leaks. Example logging, URL catching, analytics data sent to third parties, etc.

- Data logging: Logging is a technique that is used by developers to debug while developing an application. Centralized logging is provided by the android by the Log API using the logcat command. These logs contain information on what all events are taking place on a device but it should not contain any private information of the user.
- Analytics data sent to third parties: Some apps are designed in a way that allows the use of third-party API which may access information like the location of the device etc.
- URL catching: Some of the web view-based applications have an issue of leaked URL which allows attackers or hackers to access the browsing history, cached data and in some cases, the hacker may even get to the extent of taking the complete session of the user.

The threat model of Android can be illustrated concerning the different layers of Android architecture as shown in Figure 2. It shows how a layer in Android open stack architecture can be susceptible to threat. These threats have been exploited by attackers in the past and the attack taxonomy is presented in detail in the following section. To present the attack taxonomy of android, the attacks at various layers of the android architecture like the kernel layer, the middleware layer consisting of the native components, libraries, and hardware abstraction layer, the application layer have been discussed. This taxonomy broken down for each layer helps understand the vulnerabilities associated with each of the layers.

3.1 Kernel Layer

The kernel is like the heart of the Android ecosystem. The kernel is responsible for all the major functionalities inside the android device which include management of device drivers, power and process management, etc. So, an attack on the kernel can tamper with the working of the complete android device. The kernel is attacked to get the root access of the device. Despite several security mechanisms like application sandboxing, cryptographic APIs, etc, certain vulnerabilities are present in the kernel layer of android. The vulnerability within the android kernel layer is the most dangerous. Taking

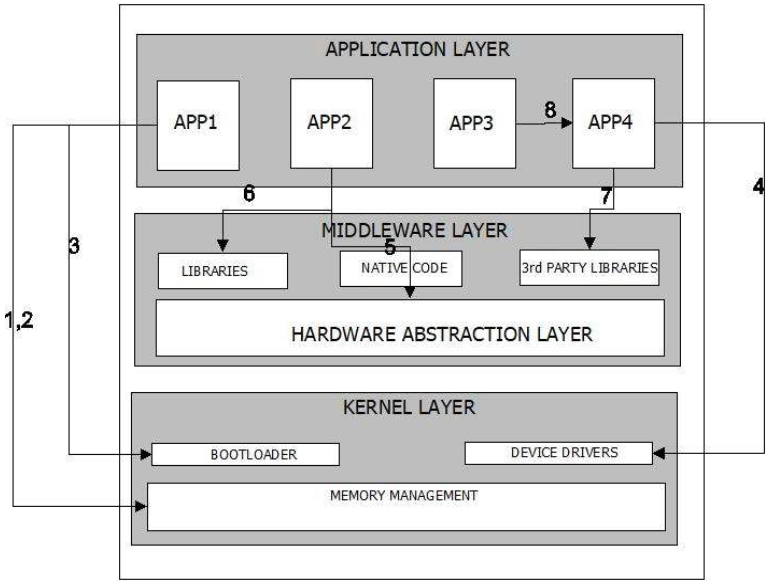

Figure 2 Android threat model.

advantage of any such vulnerability, an attacker can acquire the ability to bypass system protection mechanisms and carry out some malicious behaviour, such as gain root privilege and steal personal data [14].

Attacks Targeting Root Privilege: Attack has been shown in Figure 2 as line 1 [15–17]. There are two types of attacks that come under this category one is the Droid kungfu Attack and the other one is the Gooligan attack.

- Droid Kungfu attack: It is a malware that comes in the encrypted form in the repackaged application and it gets decrypted on getting installed in the device. It is capable of overcoming the Application sandboxing mechanism. It unlocks all the system files and functions by gaining the root privilege in the device. DroidKungfu malware uses the exploits known as RageAgainstTheCage (CVE-2010-EASY) and Exploid (CVE-2009-1185) to escalate privileges [18]. Such attacks are prevented by Security Enhanced (SE) Android present in android devices.
- Gooligan Attack: It is a malware that exploits the zero-day vulnerability (CVE-2013-6282) found in kernel API, [18] the zero-day vulnerability is called "zero-day" because it is the duration during which vulnerability is exposed and used by the attackers while there is no patch provided by

the vendor. In such kinds of attacks, some malware application is used to perform the read/write operations on the memory of the kernel.

Attacks targeting memory: Such types of attacks target memory and cause memory corruption of the device as shown in Figure 2 in line 2. In such kinds of attacks, the attackers target the bug that is present in the kernel. These loopholes or bugs are used by attackers, to add malicious content in the kernel address space or redirect the execution of the kernel to the address space where the malicious content is present [18]. There are many kinds of attacks that come under this category like the return to libc attack, jump-oriented programming attack, return-oriented programming attack, etc. [19, 20].

Attacks targeting the bootloader: Attack has been shown in Figure 2 as line 3. Bootloader [21] is the piece of code that takes care of what all applications are needed to boot up so that the device runs properly when switched on. It is also responsible for the security of the device by ensuring the integrity of the chain of trust policy [18]. According to the CoT policy, the bootloader has to verify the integrity of processes in each stage during booting before the execution of the processes. When the booting of the system takes place then the attackers find a way to get the root access by putting the malicious code, but here the security feature of the bootloader comes into play. There are different bootloaders like Huawei, Qualcomm, Nvidia having certain vulnerabilities that can be identified using BOOT STOMP [21]. CVE-2014-9798 and CVE-2015-8893 are vulnerabilities found in the Qualcomm bootloader [18].

Attacks targeting device drivers: A device driver is a way of setting up communication between hardware and software. Every hardware running on a device has a specific driver installed with it. Attacking drivers have the power of making the system unusable. Attackers can attack the driver which can lead the complete system to crash as shown in Figure 2 line 4. As soon as the new driver is released in the market the attacker starts finding the flaws in the code and uses this flaw to attack the kernel of the device. There is sufficient time between finding that flaw and fixing that flaw, the attacker makes full use of this time and launches the attack [7], such attacks are called time of check to time of use (TOCTOU) attacks.

3.2 Middleware Layer

This is the layer just above the kernel layer. It consists of the android runtime and the native components layer, which further consists of a hardware abstraction layer (HAL) and all the libraries of android.

Attack targeting HAL: Hardware abstraction layer consists of the interface of various applications like GPS, Bluetooth, Wi-Fi, etc. A key reinstallation attack is one such attack. One mandatory condition for executing such an attack is that the attacker should be in the Wi-Fi range of the device. Such kind of attack as shown in Figure 2 line 5, can cause two things one is that it can steal the personal data and the other is that it can decrypt the encrypted files as well. Another kind of attack is a brute force attack it aims at stealing all the private passwords of the system. A brute force attack can be prevented by incorporating strong encryption algorithms and the Key Derivation Process (KDP) employing the SCRYPT function for generating a key-based password [18].

Another attack that targets the drivers of the android system is the Invisible man attack. It can even go to the extent of stealing the bank details. It gets into the system when the user installs fake Flash updates, disguised as authenticated ones, from the unauthorized website [18]. In the recent versions of the android, the security has been improved by providing a separate hardware abstraction layer for every driver present in the android. It helps in limiting the privileges and grants only those permissions which are necessary for the execution of an application.

Attack targeting libraries and native components: Libraries are the most important part of the android application package file. Most of the libraries or the core libraries present in the android system are written in C/C++ [22, 23]. Some of them are even responsible for maintaining the security of the android system but none of them is error-free. Attackers find these bugs in the libraries and launch an attack on the android system as shown in Figure 2 line 6. All these bugs and errors are fixed in the updated versions of the libraries.

Attacks targeting third-party libraries: Some faults take more than usual time to get solved which leads to really serious security threats as that time is utilized by the user to perform the attack on the device [18, 24]. In 2016, buffer overflow vulnerability was reported in the getaddrinfo() function; this function was used to search domain names included in the GNU C library. The solution to the above-mentioned problem took 7 months during which attackers caused severe damage by the attacks. Most of the attacks take place in this layer because some unnecessary permissions are granted by the user during installation time, some information is so dangerous that once given will provide open access to the complete system. Such an attack has been shown in Figure 2 as line 7.

Many detection tools are present for the detection of attack in the system but attackers can easily surpass these tools by using code obfuscation techniques. Some of the popular tools are WHYPER [25], LibSift, LibScout, etc. [18] WHYPER is a tool that uses NLP for detection, it has a dataset of 581 applications and an outstanding accuracy of 97.3%.

3.3 Application Layer

Hackers are capable of finding a way to attack any kind of application whether it is preinstalled, installed using Google play store, or installed using any third-party source [26]. Some attackers gain root access to the system and bring the system to a standstill. Some attacks steal private information and sell it for money or use it to collect the ransom. Attacks that target the applications mostly cause over usage of memory, battery, and processing power. The applications that are installed from third-party sources are the most dangerous ones. Permissions are asked from the user while installing an application when an application is installed from a source other than google play store. Then there are certain permissions which are not required for an application to run but can be asked from the user which the user allows but can lead to the execution of malicious activity in the background while the application is running, such as stealing personal information by acquiring the permission of accessing the messages which are not required to run an application.

Runtime Information Gathering Attack (RIG) is an attack, as shown in Figure 2 line 8, through which attackers can accumulate information from the victim's system, during the run-time of the malicious application [18]. This attack taxonomy provides a clear view of the different types of attacks that are possible in the Android system. Having studied these attacks layer-wise makes it easier to understand the vulnerabilities that are possible in these layers. Also, it allows the analyst to study the different artifacts that may be exploited by the attacker. These artifacts help to study the features that can later be used in the detection of malware. The researchers further study these detection approaches in the following sections.

4 Malware Detection System

In this section, the malware detection approaches have been discussed. These approaches are classified as static, dynamic, and hybrid approaches based on the type of feature being used for detection. Further, the reverse engineering

tools that have been proposed in past research that aid in the detection of malware have also been discussed. Two case studies for extracting features from APKs using the tool MOBSF are also done and their results are also evaluated. To enable the reader to understand the use of features, the machine learning algorithms that classify the data into malware or benign have been explained. Their performance parameters are listed so that the analyst can compare these algorithms and choose the better out of these.

4.1 Detection Approaches

The malware detection approaches have evolved over the years. While most research studies discuss the detection mechanisms, in [27], the use of prediction technologies using the cyber kill chain and industrial control system to reduce cyber-based attacks has been explained. This article discusses the detection approaches for Android malware analysis. From the signature-based techniques now the behaviour-based, deep learning-based approaches are being studied to provide an efficient and faster detection mechanism [28]. In [28], Aslan et al. have presented the malware detection approaches that can be divided into broad categories viz. static and dynamic. The detection approach can further be classified using the features that are obtained from these analyses. It may be a signature-based or cloud-based or behaviour-based approach which can be decided by choosing the feature obtained after the application has been statically or dynamically decomposed to reveal its features. In [29], Wang et al., have also discussed the static, dynamic, and hybrid approaches and have explained each of the features that are obtained from one of these techniques. In the section that follows the different detection approaches and the features that can be revealed using these techniques have been studied.

Static Analysis – Static analysis techniques were the most favoured techniques for detection purposes such as signature matching. The static analysis techniques work based on extracting several features from the artifacts present in the application. E.g. in an apk the different files and folders present in the apk can be used to reveal the features. The researcher can obtain opcodes, strings, permissions, etc from the different files. The process of static analysis includes decompressing the apk first to reveal its complete structure. This will include folders and files. The files such as Manifest.xml are very important to extract the features related to permissions, intents, activities, etc. The following text presents a discussion on research works using the static features [29] such as API class, opcodes, network addresses,

Intent, intent filter, permissions, etc. In [30], Arora et al. have discussed identifying malware by using the permissions in pairs. They compare graphs built using different benign and malware dataset using the permissions pairs. In [31], Feng et al. have also used permissions over the neural network for detection.

In [32], Liu et al. have used the apriori algorithm to identify the permission combinations and then ranking them in order of threat before feeding the feature vector to the classification algorithms. In [33], Xu et al. has also considered using Inter Component Communication (ICC) between the apps as the means for detection They consider all of the Intent, intent-filter, and app components while performing the classification. In [34], Octeau et al. have used probability modeling using the ICC. The components are linked through ICC and these links are weighted based on probability modeling. In [35], Jerlin et al. have proposed the use of API Calls as a method of detection. They first identify the upper and lower boundary for the values of the features and then applying the Rete algorithm for generating rules to detect malware. A multi-dimensional naïve Bayes algorithm is later used or classification. In [36], Garcia et al. have used API and identified them into categories such as package level and method level. To increase accuracy, they have also considered reflection and native code as a feature while performing detection.

In [37], Yong et al. propose to use comparison tools to identify the changes between the source code and the new code. Then from this change point build the data flow and from syntax, the analysis gets the statement block tree. From analyzing the new code also a function call path can be generated. These all are analyzed to get the data change path. In [38], Nguyen et al. have considered a special type of CFG called lazy binding CFG. They first propose to convert the lazy binding CFG to images and then using deep learning they classify the apk into malware or benign.

Dynamic Analysis – Dynamic Analysis is preferred more nowadays as the malware writers are smartly hiding the malicious code in the code using obfuscation and packaging. The malware has evolved over the years and now the variety of malware includes oligomorphic, metamorphic, polymorphic, packaged, etc. These can easily evade the static analysis techniques but reveal their true nature when subjected to dynamic analysis. An apk is executed either in an emulator or on a real device and all events/actions related to it are captured through logs. These logs are then used to obtain the features that are used for further detection. The following text discussed the dynamic features such as [29] System Calls, Network features, API calls, used permissions,

Table 1 Features, algorithms, and datasets

Features	Machine Learning Algorithms	Dataset
API	1. Random Forest, 1NN,3NN, SVM [42] 2. SVM,kNN, Random Forest [43] 3. C4.5, DNN, LSTM [44]	1. Drebin, Virusshare 2. Google Play, Anzhi, LenovoMM, Wandoujia 3. Benign – Androzoo Malware – AMD
Permissions	1. NLP (Bag-of-words, Vector space model) Naïve Bayes, Logistics regression, SVM, NB-SVM [45] 2. SVM, ID3, RF, NN, KNN and bagging [46]	1. Google Play 2. StormDroid
Intent	1. SVM [33]	1. Google Play, Drebin
Opcode	1. MLP [47] 2. Manhattan distance, BPNN, SVM, CNN, SVM [48]	1. Google Play, Drebin 2. Google Play, Drebin

SMS Event, Phone events, Battery feature, User Interaction. In [39], Das et al. have used system calls as a feature to identify malware. They use *strace* and monkey tool to execute the app and get the logs from which the system calls are obtained. These are then normalized using tf-idf or some frequency measure which are then fed to classification algorithms. In [40], Turrisi Da Costa et al. propose to use system calls to identify mobile botnets.

Hybrid Analysis – Using any single approach is not as efficient as the combination approaches for detection. Disadvantages in each of static and dynamic analyses led to a new approach called hybrid which works by using both in different ways. Several hybrid-based approaches [41] make use of static analysis followed by dynamic analysis to get complete coverage. Sometimes Dynamic analysis aided by static analysis also combines to give another hybrid approach. Some of the features that are used for analysis have been given in Table 1. Table 1 also lists the Machine Learning algorithms used and the dataset used for these research papers.

4.2 Reverse Engineering Tools

The Malware detection methods have been broadly classified as static, dynamic, and hybrid as also discussed in the previous section. Several tools

have emerged over the years to aid in these types of techniques. This section lists and reviews some of the tools that have been shown as great importance in the detection of malware.

APKtool [49] – A very common reverse engineering tool that helps to decompress an apk. The tool can decompile the apk into files and folders which can be analyzed statically. Androguard [50] – Another tool that aids in static analysis is androguard. It is built using Python and can work with different types of android resources such as dex files, apk files, XML files, etc to retrieve features from these files. FlowDroid [51] – This tool helps in performing the taint analysis of an application. The tool helps to identify the flow of the data from a source to a sink. This ensures that no vulnerability is left out while performing the analysis. Android Tamer [52] – It is like an OS for Android for performing the malware analysis. It is a VM that comes with several inbuilt tools that aid in malware detection. DroidBox [53] – DroidBox is for dynamic analysis of an app. It helps to generate the logs when an apk is run with it. It performs several actions on its own so that different dynamic features such as phone and SMS events, user interaction can be captured and studied later.

In the following sections, the implementation of these tools is studied so that a better understanding of the features and tools can be made available.

4.3 Case Study on Feature Extraction

This section discusses a very important step concerning the detection process. The detection algorithms work on the features of the different artifacts of an apk. These features lay the foundation stone for the detection and classification of an incoming application. As discussed in the previous sections, static and/or dynamic analysis is performed on the apk to obtain the features. Where static features can be obtained by extracting some signification strings from the files, dynamic analysis requires that the code is executed and behavior is observed. This is mostly done on an emulator so that its environment can be controlled.

Here, the process of static analysis using tools such as Android Tamer, and MOBSF [53] has been discussed. Android Tamer is a Virtual Machine that consists of several inbuilt tools. MOBSF is one such tool that comes bundled with the Android Tamer. It can be used for both static and dynamic analysis of an apk. For static analysis, upload the apk in the user interface of MOBSF. The tool performs operations on the apk to list out all the features of the apk such as permissions, components, APIs, code nature, etc. The experiments

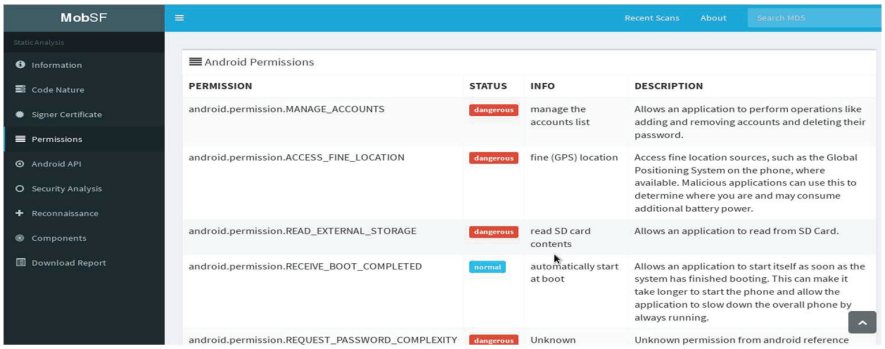

Figure 3 Permission features.

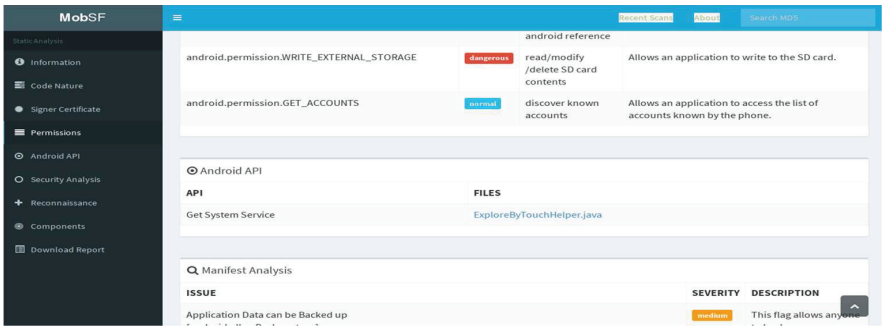

Figure 4 API feature.

were performed for a test apk using MOBSF. Figure 3 shows the permission features. The tool MOBSF will list out all the permissions that have been defined for the app in question. MOBSF also categorizes the permission in categories such as dangerous or normal so that the analyst can point out if the app is using any dangerous permissions and flag it as a possible malicious app. Figure 4 lists the APIs.

The details such as the filename in which the API call is being made are also listed. Figure 5 contains the Components details such as activities, services, etc. It includes all the filenames along with package names of the files for each of these categories. Figure 6 lists the entries in the manifest file which also contains the description related to the Manifest entries. As shown in Figure 6, the LaunchActivity has an alias that can be shared with other apps. These can also be downloaded as a report for further analysis. Once these features have been obtained, analysts can proceed with the classification

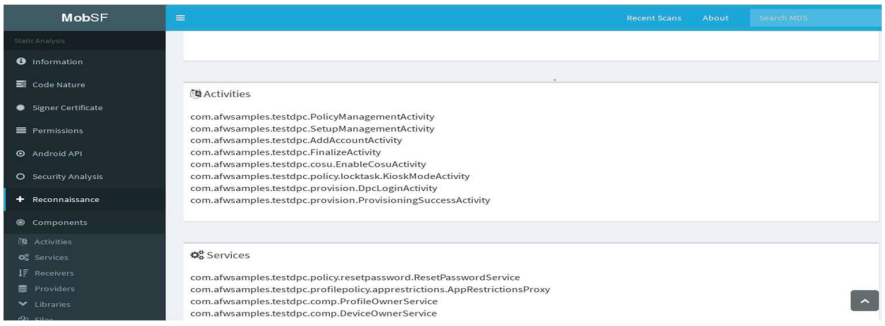

Figure 5 Component feature.

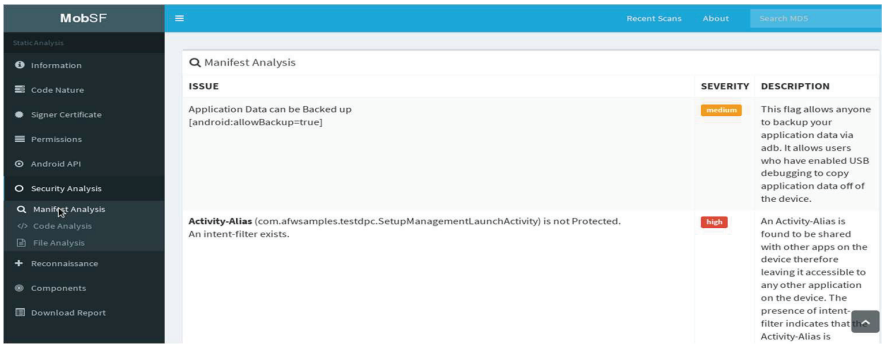

Figure 6 Manifest features.

of the file based on its features using machine learning techniques. This has been discussed in the next section.

4.4 Case Study on Machine Learning Classification

Machine Learning techniques are being used as a quick and efficient means of malware detection. Machine learning techniques can be classified into three broad categories: Supervised, unsupervised, and reinforcement learning. These categories are further categorized as supervised – regression-based and classification based. Unsupervised consists of clustering-based techniques. Several algorithms for each of these techniques are available e.g. logistic regression, naïve Bayes, Decision tree. Using ensemble learning the performance/accuracy of the classification models can be enhanced. The different ensemble learning techniques are bagging, boosting, and voting.

Table 2 Classification results (in %)

Algorithm	TPR	FPR	Precision	F-Measure	ROC
J-48	97.3	3.2	97.3	97.3	97.5
AdaBoost1(J48)	98.7	1.7	98.7	98.7	99.8
Bagging(J-48)	98.0	2.6	98.0	98.0	99.6
Voting(J48)	97.3	3.2	97.3	97.3	97.5

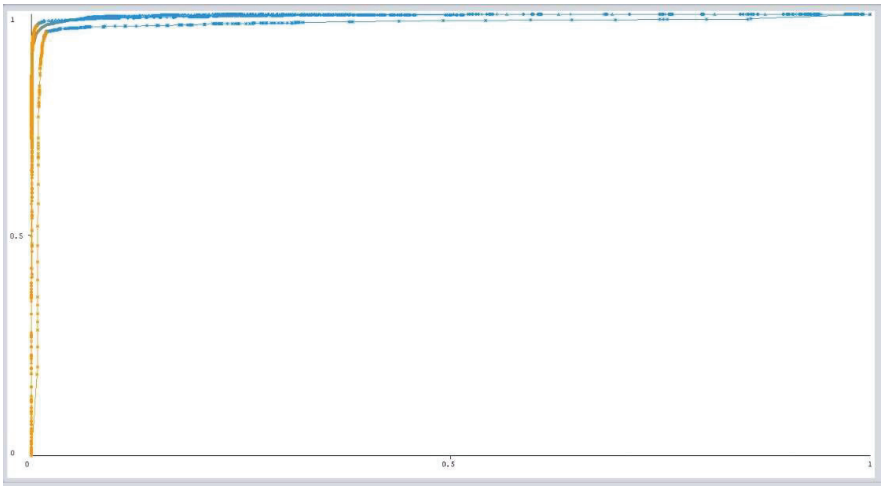

Figure 7 Threshold curve.

As a part of demonstrating the implementation process using machine learning, the dataset [55] consisting of 215 features has been used. The features include API calls, Intent, permissions, etc. The experiments were conducted in the Weka tool using 10-fold cross-validation. The first classifier used is J48 which is a decision tree. Table 2 below shows the results thus obtained. The J-48 performance is improved by using ensemble techniques such as AdaBoost and bagging. The voting technique, however, makes no change to the results.

The following metrics were considered for evaluating the performance of the models

- TPR – is the True positive rate also called sensitivity or recall. It is the number of actual positives correctly identified. Its value is TP/(TP+FN)
- FPR – is the False Positive rate and is the ratio of incorrectly classified instances to actual negatives. Its value is FP/(FP+TN)

- Precision – is the number of positives that belong to the positive class.
- F-Measure – helps combine the result of precision and recall. Its formula is (2*precision*recall)/ (precision +recall)
- ROC – Receiver Operating characteristics is the performance measure and is a probability curve.

Figure 7 is the threshold curve for all the four different algorithms chosen for this study. The minimum AUC-ROC (Area under Curve – Receiver Operating Characteristics) shown is for the J48. The other ensemble algorithms show a better value. The AUC-ROC curve is to measure the performance of a classification model for different threshold values. The AUC is the area under the curve and the higher the value of AUC the better is the classification model.

5 Related Work

With the advent of the Internet and increased usage of smartphones and mobile devices, the rise in attacks on these devices has also been significant. The attackers are finding new ways to compromise the device to either sell or maliciously use the device user's information. Intrusion detection in Android is an emerging field and several research topics have tried to understand the complexities involved in identifying an efficient system for detection. The researchers have explored many analysis approaches in conjunction with the latest technologies to be able to come up with a system that not only identifies malware but also classifies it into its respected families.

Several survey papers cover the details about attacks on Android. Bhat et al. [18] have presented the different possible attacks on Android concerning its architecture. All the details related to attacks and possible defenses of the same have been discussed. They also discuss techniques that can help in preventing attacks through the use of tools. They have also discussed the malware analysis approaches categorized as static, dynamic, and hybrid. Faruki et al. [9] have also discussed android security in their paper. They have presented an in-depth analysis of the Android internal structure explaining the different components and security mechanisms. They have discussed security issues in Android including a detailed review of the different types of malware. For this, they have also given a chronological description of the malware. Static and dynamic approaches have also been discussed and several tools and their techniques are presented in detail.

Aslan et al. [28] in their paper have classified the malware analysis approaches into signature-based, behavior-based, heuristic-based, model-checking-based, deep learning-based, cloud-based, mobile-based, IoT-based. For each of these classifications, they have presented corresponding techniques and their detailed evaluation. Another paper that discusses the malware analysis is by Wang et al. [29] in which the researchers have performed an in-depth analysis of the features for each of static, dynamic, and hybrid approaches.

The malware analysis can be categorized based on the features obtained from the application. The features obtained without executing the app are the static features and those obtained by executing the app are dynamic features. Several pieces of research focus on the usage of a single or combination of features. Wang et al. [56] have used locally sensitive API calls to perform malware analysis in their research. The accuracy and recall rate they obtain for classification of malware into families is as high as 0.96. Ma et al. [57] in their paper have also used API calls to perform malware analysis. They use the API calls, their frequency, and the sequence to construct an ensemble method and provide comparisons of this ensemble method with the individual model's performance.

Using permissions as a feature for malware analysis Alswaina et al. [46] have presented an approach in which they identify a reduced set of permissions using an extremely randomized tree. These permissions are then represented using binary values and weighted values, results of which are compared to achieve a better model. Arora et al. [30] have also presented their approach using permissions, they have however used permissions in pairs. They have identified different permission pairs that can be classified as dangerous and based on these graphs are constructed for different datasets which are then compared with malware datasets to identify malware. In [58] Zhang et al. have shown the use of op-code for malware analysis. They proposed to construct a Dalvik opcode graph using weighted probability. This graph is then reduced so that the complexity can be reduced. The similarity between the existing and new graphs is used to label the new one as malicious or not.

This paper focuses on reviewing the malware analysis approaches by analyzing the threat model and the different types of attacks. This article also shows the usage of the features utilizing a case study in which the different machine learning algorithms are implemented on a dataset. This article also shows the feature extraction techniques using the tool MOBSF which reveals

both static and dynamic features of an application. Such case studies have not been presented in other research papers.

6 Future Directions

The Android OS continues to get evolve fixing the vulnerabilities reported by releasing the patches but still malware writers keep exploiting any small loophole they can find. Thus, the research in this area continues to grow by looking for new avenues of improvement at all the dimensions of the Android. Some of the future directions that need to be studied are

- Cloud Context – The mobile devices are resource-constrained and thus need to use the cloud to enable the huge computing requirements at users' end. The introduction of cloud brings in threats related to the cloud environment that seems to be a promising research direction.
- Efficient and faster results – Another direction that is possible is for a faster and result-oriented solution for malware detection. Possible use of deep learning also in cloud context and combinational approaches is required to develop such a system.
- Deep learning implementation – Machine learning (ML) algorithms have disadvantages related to time consumption. This cane be improved by use of deep learning-based algorithms where the time consumption for the analysis is less as compared to ML techniques. These algorithms need to be studied in depth to come up with efficient malware analysis.
- Stronger and secure algorithms – malware writers are smart enough to understand the detection process and thus write newer malwares that can evade these algorithms. There is a need to secure the algorithms so that these can not be broken for safer environment.

7 Conclusion

In this article, the Android system by presenting a detailed study on the Android architecture and the attack taxonomy of Android for each of its layers has been reviewed. To enable the readers to get a better understanding of how the Android system is susceptible to threat from the outside world a threat model has been proposed. The attack taxonomy and threat model will ensure a better understanding of the possible vulnerabilities in Android. The detection approaches and the different tools that aid these techniques have been discussed. To provide a better understanding, case studies illustrating

the feature extraction mechanism using the tools and the machine learning implementation for the detection and classification are presented. Some of the gaps identified during this study are also given as future directions to enable further research in those areas.

References

[1] P. Kaur and S. Sharma, "Google Android a mobile platform: A review," 2014 Recent Adv. Eng. Comput. Sci. RAECS 2014, pp. 6–8, 2014, doi: 10.1109/RAECS.2014.6799598.

[2] "Statista". [Online] Available: https://www.statista.com

[3] Y. Kim, T. Oh, and J. Kim, "Analyzing User Awareness of Privacy Data Leak in Mobile Applications," *Mob. Inf. Syst.*, vol. 2015, 2015, doi: 10.1155/2015/369489.

[4] D. Wu, D. Gao, E. K. T. Cheng, Y. Cao, J. Jiang, and R. H. Deng, "Towards understanding android system vulnerabilities: Techniques and insights," *AsiaCCS 2019 – Proc. 2019 ACM Asia Conf. Comput. Commun. Secur.*, pp. 295–306.

[5] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen, "A survey on mobile malware detection techniques," *IEICE Trans. Inf. Syst.*, vol. E103D, no. 2, pp. 204–211, 2020, doi: 10.1587/transinf.2019INI0003.

[6] H. Li, D. Zhan, T. Liu, and L. Ye, "Using Deep-Learning-Based Memory Analysis for Malware Detection in Cloud," *Proc. – 2019 IEEE 16th Int. Conf. Mob. Ad Hoc Smart Syst. Work. MASSW 2019*, pp. 1–6.

[7] N. Xie, X. Wang, W. Wang, and J. Liu, "Fingerprinting Android malware families," *Front. Comput. Sci.*, vol. 13, no. 3, pp. 637–646, 2019, doi: 10.1007/s11704-017-6493-y.

[8] J. B. Hur and J. A. Shamsi, "A survey on security issues, vulnerabilities and attacks in Android based smartphone," *2017 Int. Conf. Inf. Commun. Technol. ICICT 2017*, vol. 2017-December, pp. 40–46.

[9] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic android malware detection system with ensemble learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018, doi: 10.1109/ACCESS.2018.2844349.

[10] P. Faruki et al., "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015, doi: 10.1109/COMST.2014.2386139.

[11] Krajci I., Cummings D. (2013) History and Evolution of the Android OS. In: Android on x86. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-6131-5_1.

[12] Sufatrio et al., "Securing Android: A Survey, Taxonomy, and Challenges," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–45, 2015, doi: 10.1145/2733306.
[13] S. Hutchinson and D. C. Varol, "A Survey of Privilege Escalation Detection in Android," *2018 9th IEEE Annu. Ubiquitous Comput. Electron. Mob. Commun. Conf. UEMCON 2018*, pp. 726–731.
[14] H. Meng, V. L. L. Thing, Y. Cheng, Z. Dai, and L. Zhang, "A survey of Android exploits in the wild," *Comput. Secur.*, vol. 76, pp. 71–91, 2018, doi: 10.1016/j.cose.2018.02.019.
[15] B. Kong, Y. Li, and L.-P. Ma, "PtmxGuard: An Improved Method for Android Kernel to Prevent Privilege Escalation Attack," *ITM Web Conf.*, vol. 12, p. 05010, 2017, doi: 10.1051/itmconf/20171205010.
[16] L. Nguyen-Vu, N. T. Chau, S. Kang, and S. Jung, "Android Rooting: An Arms Race between Evasion and Detection," *Secur. Commun. Networks*, vol. 2017, no. 4, 2017, doi: 10.1155/2017/4121765.
[17] Y. an Tan *et al.*, "A root privilege management scheme with revocable authorization for Android devices," *J. Netw. Comput. Appl.*, vol. 107, pp. 69–82, 2018, doi: 10.1016/j.jnca.2018.01.011.
[18] B. Soewito and A. Suwandaru, "Android Sensitive Data Leakage Prevention with Rooting Detection Using Java Function Hooking," *J. King Saud Univ. – Comput. Inf. Sci.*, no. xxxx, 2020, doi: 10.1016/j.jksuci.2020.07.006.
[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, 2019, doi: 10.1145/3301285.
[20] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed," no. May, pp. 1–17, 2019.
[21] D. Gruss *et al.*, "Page cache attacks," *Proc. ACM Conf. Comput. Commun. Secur.*, pp. 167–180.
[22] N. Redini *et al.*, "Bootstomp: On the security of bootloaders in mobile devices," *Proc. 26th USENIX Secur. Symp.*, pp. 781–798.
[23] R. Fedler, M. Kulicke, and J. Schütte, "Native Code Execution Control for Attack Mitigation on Android," pp. 15–20.
[24] J. Zhang, A. R. Beresford, and S. A. Kollmann, "LibiD: Reliable identification of obfuscated third-party android libraries," *ISSTA 2019 – Proc. 28th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, pp. 55–65.
[25] M. Fan, X. Luo, J. Liu, C. Nong, Q. Zheng, and T. Liu, "CTDroid: Leveraging a Corpus of Technical Blogs for Android Malware

Analysis," *IEEE Trans. Reliab.*, vol. 69, no. 1, pp. 124–138, 2020, doi: 10.1109/TR.2019.2926129.

[26] L. Davi, A. Dmitrienko, A. R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6531 LNCS, pp. 346–360, 2011, doi: 10.1007/978-3-642-18178-8_30.

[27] R. Kour, A. Thaduri, and R. Karim, "Railway defender kill chain to predict and detect cyber-attacks," *J. Cyber Secur. Mobil.*, vol. 9, no. 1, pp. 47–90, 2020, doi: 10.13052/JCSM2245-1439.912.

[28] O. Aslan and R. Samet, "A Comprehensive Review on Malware Detection Approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020, doi: 10.1109/ACCESS.2019.2963724.

[29] W. Wang et al., "Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019, doi: 10.1109/ACCESS.2019.2918139.

[30] A. Arora, S. K. Peddoju, and M. Conti, " PermPair : Android Malware Detection Using Permission Pairs ," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1968–1982, 2019, doi: 10.1109/tifs.2019.2950134.

[31] Y. Feng, L. Chen, A. Zheng, C. Gao, and Z. Zheng, "AC-Net: Assessing the Consistency of Description and Permission in Android Apps," *IEEE Access*, vol. 7, pp. 57829–57842, 2019, doi: 10.1109/ACCESS.2019.2912210.

[32] Z. Liu, Y. Lai, and Y. Chen, "Android malware detection based on permission combinations," *Int. J. Simul. Process Model.*, vol. 10, no. 4, pp. 315–326, 2015, doi: 10.1504/IJSPM.2015.072522.

[33] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-Based Malware Detection on Android," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 6, pp. 1252–1264, 2016, doi: 10.1109/TIFS.2016.2523912.

[34] D. Octeau et al., "Combining static analysis with probabilistic models to enable market-scale android inter-component analysis," *ACM SIGPLAN Not.*, vol. 51, no. 1, pp. 469–484, 2016, doi: 10.1145/2837614.2837661.

[35] M. A. Jerlin and K. Marimuthu, "A New Malware Detection System Using Machine Learning Techniques for API Call Sequences," *J. Appl. Secur. Res.*, vol. 13, no. 1, pp. 45–62, 2018, doi: 10.1080/19361610.2018.1387734.

[36] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-Resilient detection and family identification of android malware," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 3, pp. 1–29, 2018, doi: 10.1145/3162625.

[37] C. Yong, M. Yongmin, and S. Meie, "Data change analysis based on function call path," *Int. J. Comput. Appl.*, vol. 40, no. 3, pp. 1–10, 2018, doi: 10.1080/1206212X.2017.1413625.

[38] M. H. Nguyen, D. Le Nguyen, X. M. Nguyen, and T. T. Quan, "Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning," *Comput. Secur.*, vol. 76, pp. 128–155, 2018, doi: 10.1016/j.cose.2018.02.006.

[39] P. K. Das, A. Joshi, and T. Finin, "App behavioral analysis using system calls," *2017 IEEE Conf. Comput. Commun. Work. INFOCOM WKSHPS 2017*, pp. 487–492.

[40] V. G. Turrisi Da Costa, S. Barbon, R. S. Miani, J. J. P. C. Rodrigues, and B. B. Zarpelão, "Mobile botnets detection based on machine learning over system calls," *Int. J. Secur. Networks*, vol. 14, no. 2, pp. 103–118, 2019, doi: 10.1504/IJSN.2019.100092.

[41] Y. Liu, K. Guo, X. Huang, Z. Zhou, and Y. Zhang, "Detecting Android Malwares with High-Efficient Hybrid Analyzing Methods," *Mob. Inf. Syst.*, vol. 2018, 2018, doi: 10.1155/2018/1649703.

[42] L. Onwuzurike, E. Mariconti, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version)," vol. 22, no. 2, 2017.

[43] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018, doi: 10.1109/ACCESS.2018.2835654.

[44] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019, doi: 10.1109/ACCESS.2019.2896003.

[45] Y. Feng, L. Chen, A. Zheng, C. Gao, and Z. Zheng, "AC-Net: Assessing the Consistency of Description and Permission in Android Apps," *IEEE Access*, vol. 7, pp. 57829–57842, 2019, doi: 10.1109/ACCESS.2019.2912210.

[46] F. Alswaina and K. Elleithy, "Android Malware Permission-Based Multi-Class Classification Using Extremely Randomized Trees," *IEEE Access*, vol. 6, pp. 76217–76227, 2018, doi: 10.1109/ACCESS.2018.2883975.

[47] Stuart Millar, Niall McLaughlin, Jesus Martinez del Rincon, Paul Miller, and Ziming Zhao. 2020. DANdroid: A Multi-View Discriminative

Adversarial Network for Obfuscated Android Malware Detection. In Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20). Association for Computing Machinery, New York, NY, USA, 353–364.

[48] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik Opcode Graph Based Android Malware Variants Detection Using Global Topology Features," *IEEE Access*, vol. 6, pp. 51964–51974, 2018, doi: 10.1109/ACCESS.2018.2870534.

[49] "APKtool" [Online] Available: https://ibotpeaches.github.io/Apktool/

[50] "Androguard" [Online] Available: https://github.com/androguard/androguard

[51] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, pages 259–269, New York, NY, USA, 2014. ACM

[52] "Android Tamer". [Online] Available: https://github.com/AndroidTamer/Tools.

[53] "MOBSF" [Online] Available: https://mobsf.github.io/docs/#/

[54] P. Lantz, "An Android Application Sandbox for Dynamic Analysis," 2011.

[55] Yerima, Suleiman (2018): Android malware dataset for machine learning 2. figshare. Dataset. https://doi.org/10.6084/m9.figshare.5854653.v1

[56] W. Wang, J. Wei, S. Zhang, and X. Luo, "LSCDroid: Malware Detection Based on Local Sensitive API Invocation Sequences," *IEEE Trans. Reliab.*, vol. 69, no. 1, pp. 174–187, 2020, doi: 10.1109/TR.2019.2927285.

[57] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019, doi: 10.1109/ACCESS.2019.2896003.

[58] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik Opcode Graph Based Android Malware Variants Detection Using Global Topology Features," *IEEE Access*, vol. 6, pp. 51964–51974, 2018, doi: 10.1109/ACCESS.2018.2870534.

Biographies

Charu Negi is currently working as Assistant Professor in Graphic Era Hill university, Dehradun, India. She is a research scholar working under the guidance of Dr Preeti Mishra, from Graphic Era Deemed to be University, Dehradun, India. Her research interests include Android Security, Malware detection, Machine Learning.

Preeti Mishra is currently working as an Associate Professor in Graphic Era Deemed to be University, Dehradun, India. She has been awarded Ph. D. in Computer Science and Engineering from Malaviya National Institute of Technology Jaipur, India under the supervision of Dr. Emmanuel S. Pilli and Prof. Vijay Varadharajan (2017). She has been a Visiting Scholar in Macquarie University, Sydney, Australia in 2015. She is an active IEEE member and her interest includes Cloud Security, Cyber Security and Machine Learning, android Security.

Pooja Chaudhary is a BTech student at Graphic Era deemed to be University at Dehradun since summer 2018. She has worked on research papers like detecting rice leaf disease using Image Processing and Machine Learning which was published in 2020. She served as the vice technical head of GEU ACM from March 2019–July 2020. She is currently in the 3rd year of her graduation and wishes to learn and build technologies that impact the world positively in the future.

Harsh Vardhan, is a B-Tech student at Graphic Era Deemed to be University since summer 2018. He has researched on detection of diseases in rice leaves using image processing and has a research paper about it which was published by Springer in 2020. He has keen interest in data structures and algorithms and is working on improving his problem solving skills. He is currently in his penultimate year of graduation and is working towards learning and contributing more to various fields of computer science.