Received: 28 August 2019 | Revised: 15 February 2021 | Accepted: 25 February 2021

DOI: 10.1002/cpe.6272

RESEARCH ARTICLE

WILEY

A survey on analysis and detection of Android ransomware

Shweta Sharma[1] | **Rakesh Kumar**[2] | **C. Rama Krishna**[1]

[1]Department of Computer Science and Engineering, National Institute of Technical Teachers Training and Research, Chandigarh, India

[2]Department of Computer Science and Engineering, Central University of Haryana, Haryana, India

Correspondence
Shweta Sharma, Department of Computer Science and Engineering, National Institute of Technical Teachers Training and Research, Chandigarh 160019, India.
Email: shweta.rt.sharma@gmail.com

Abstract

Smart-phones have become a necessity for users due to their abundance of services such as global positioning system, Wi-Fi, voice/video calls, SMS, camera, and so forth. It contains personal information of users including photos, documents, messages, and videos. Android-based smart-phones enriched with many applications (commonly known as apps) fascinates users to use this ubiquitous technology up to a full extent. With open architecture and 73% of market share, Android is the most popular mobile operating system (OS) among developers. At the same time, the increasing popularity of Android OS woos attackers or cyber-criminals to exploit its vulnerabilities. The attackers write malicious code to harm the device and grab users' sensitive information. For example, ransomware (a form of malware) demands ransom from victims to liberate the ceased material for illegal financial gain. The existing survey papers cover the analysis and detection of generic Android malware. The focus of this survey paper is to present an in-depth threat scenario of Android ransomware. This article not only provides a comprehensive survey on analysis and detection methods for Android ransomware since its beginning (2015) till date (2020); but also presents observations and suggestions for researchers and practitioners to carry out further research.

KEYWORDS

Android package kits, Android security, deep learning, machine learning, malware, ransomware

1 | INTRODUCTION

Android malware is a malign app or an annoying program code that performs malicious actions to harm the Android-based smart-phone devices without the owner's consent.[1] Malware can spoil the Android device and stop its functions by stealing data, inserting malicious data, deleting data, modifying data, hiding data, and so forth. Android malware includes ransomware (to lock the phone screen and/or encrypt the data), grayware (used by companies to collect users data for marketing or creating own datasets), spyware (to keep an eye on users actions and collect personal information), and madware (mobile adware to bombard users phone with malicious apps).

Android ransomware is a type of malware that demands ransom from victims to liberate the seized material of Android-based smart-phone devices.[2] There are three types of analysis methods for feature extraction of Android ransomware: static, dynamic, and hybrid. The static analysis method extracts features by analyzing the program code of ransomware without running it in a virtual environment. The dynamic analysis method extracts run-time features by executing the program code of ransomware in a virtual environment. The hybrid analysis method implements both static and dynamic analysis methods. After feature extraction, the detection methods include machine learning algorithms to stay one step ahead of cyber-criminals while detecting ransomware attacks on Android operating system (OS).

Most of the existing survey papers in literature[2-7] includes a study of research papers for detection of generic Android malware. However, this survey aims to cover the threat scenario of Android ransomware and review research works done by researchers specifically to detect Android ransomware. As the first Android ransomware emerged in the year 2013;[8] following which researchers started working on the detection of Android

ransomware from the year 2015. Thus, this survey includes a significant number of research papers on Android ransomware detection within a certain time frame (2015–2020).

1.1 | Motivation

In recent years, smart-phone devices are not only used for communication purpose but fulfill professionals' objectives by writing documentation, generating PDFs, sending e-mails, and so forth. Smart-phones with 6 GB RAM include OnePlus 3T, Samsung, and HTC; while Asus ZenFone AR and OnePlus 5/6 provide 8 GB RAM to the users to run multiple applications (apps) simultaneously. Various apps such as net/mobile banking, social networking, online shopping, and online gaming are available in smart-phones to perform the same tasks as the desktop does. Out of 7.8 billion population worldwide; approximately 3.5 billion population use this ubiquitous technology.[9] However, cyber-criminals can make money more easily in smart-phones as compared with personal computer (PC) by sending SMS from victim's phone to premium rate numbers such that the mobile networking operators will debit the charging cost from victims' accounts even if the action is triggered by malware.[10] Moreover, the attackers write malicious code equipped with ransomware functionality to hack the smart-phones and demand ransom from victim to liberate the personal information present in the smart-phones including photos, documents, and more.

Over 35 million mobile malware samples have been detected by McAfee in Q4 2019.[11] The report published by Symantec antivirus[12] in the year 2019 shows that ransomware attacks have rapidly increased in comparison to other types of malware attacks on smart-phones. However, out of all mobile OS, the Android OS occupies largest market share as shown in Figure 1.[13] This increasing popularity and market share of Android OS woos cyber-criminals to write malicious apps with ransomware functionality to exploit its vulnerabilities. Unfortunately, the security solutions available for desktop (PC) cannot be applicable in smart-phone environment due to limited resources (battery, CPU) and different features (permissions, intents). This makes us write an in-depth survey on the analysis and detection of Android ransomware by adding significant contributions.

1.2 | Contributions

This article contributes to former survey papers in the following ways:

- While studying the existing survey papers, we analyze that the researchers reviewed generic Android malware. But the increasing ransomware attacks on Android-based smart-phones require a thorough study and analysis of Android ransomware. Thus, in this article, we provide a comprehensive review of the threat scenario of Android ransomware.
- There is a requirement to study the analysis methods used for feature extraction of Android ransomware. To achieve this, we discuss the features extracted by researchers by applying static, dynamic, and hybrid analysis methods on Android ransomware. We also describe files and features analyzed by the researchers followed by example papers available in the literature from the beginning (2015) till date (2020).
- It is also observed that the existing surveys provide a study on detection techniques for generic Android malware. Thus, this article discusses detection methods using machine learning techniques by giving related examples on Android ransomware.
- More experimental work is needed towards analysis and detection of Android ransomware. Thus, we present the observations and suggestions for researchers and practitioners to carry out further research on Android ransomware.

1.3 | Review methodology

We have reviewed the analysis and detection methods of Android ransomware by adopting the following steps (refer to Figure 2):

Defining research questions: Research questions are prepared after a thorough study of existing literature on Android ransomware. Table 1 shows the research questions along with the motivation. In this article, our goal is to obtain meaningful answers to these research questions.

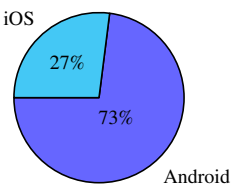

FIGURE 1 Market share of popular smart-phone operating system

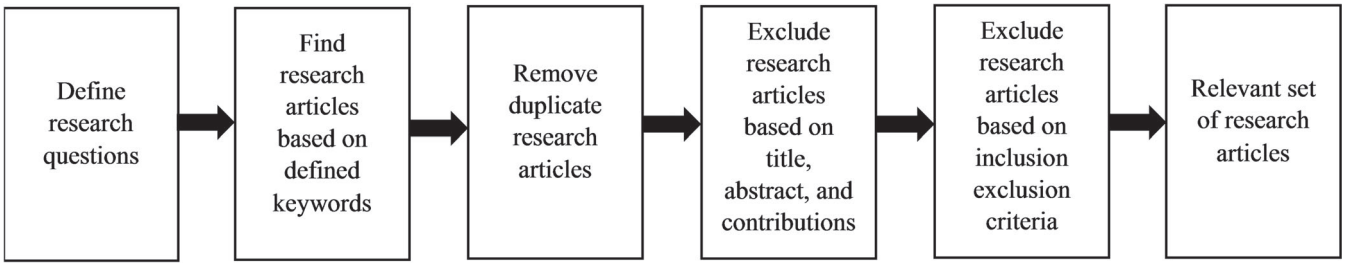

FIGURE 2 Research article review process

TABLE 1 Research questions

S. no.	Research Question	Motivation
RQ1	Which analysis method is widely adopted by the researchers for feature extraction of Android ransomware?	To compare the analysis methods, namely, static, dynamic, and hybrid for feature extraction.
RQ2	Which static and dynamic features have been extracted by the researchers after analyzing Android ransomware?	To examine the features extracted by researchers after analysis of Android ransomware.
RQ3	Which detection technique is widely adopted in the existing literature to detect Android ransomware?	To find the detection method widely used in the existing literature.
RQ4	What are the research challenges in the field of Android ransomware?	To provide observations and suggestions to fill the remaining gaps and carry out further research work.

Search strategy: To explore the landscape of Android ransomware research from its emergence (2015) till date (2020), we performed a systematic literature survey by searching the published research articles in the Google Scholar search engine. We searched the keyword "Android ransomware," "ransomware detection," and "ransomware attack on Android" in the Google Scholar search engine and found research articles published in IEEE Xplore, Springer, Wiley, ScienceDirect, MDPI, and ACM Digital Library. After collecting the research articles, duplicates are removed on the basis of title, abstract, and contributions made by the researchers. Ransomware attacks on the desktop OS (i.e., Windows, Linux) are eliminated from the survey. We reviewed a total number of 35 relevant research articles on Android ransomware that are included in our survey. Figure 3shows the percentage of the acquired relevant research articles reviewed from different sources.

The rest of the paper is arranged as follows. Section 2 gives an overview of the Android OS and security model. Furthermore, it describes the security issues related to Android OS, which gives entrance to various types of threats. Section 3 describes the threat scenario of ransomware on Android OS. Section 4 presents the analysis methods for feature extraction of Android ransomware, which act as an input to machine learning

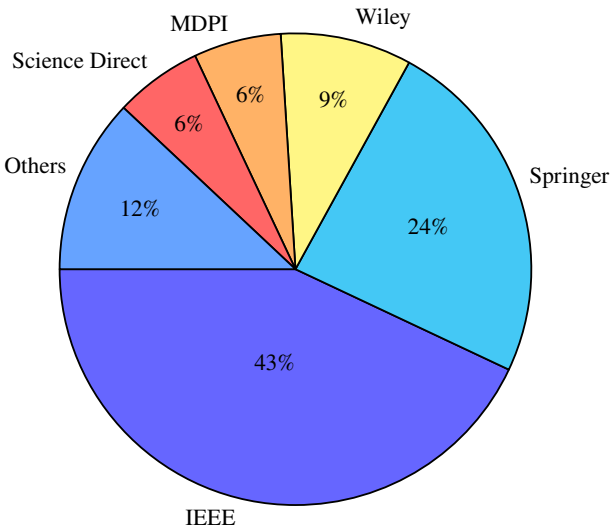

FIGURE 3 Source of surveyed Android ransomware research papers

algorithms. Section 5 describes the machine learning algorithms used by researchers to detect Android ransomware. Section 6 provides answers to the formulated research questions. Finally, Section 7concludes the paper.

2 | ANDROID OVERVIEW

Android is an open-source Linux-based OS; commercially released its first version (Android version 1.0) in 2008 and going all the way to the latest version (Android version 11) which was released on 8th September, 2020.[14] Android has overtaken other OS such as iOS, Windows phone, and Symbian to become the most popular and widely used mobile OS in the world. It supports many technologies including Wi-Fi, Bluetooth, short message service, global positioning system (GPS), accelerometers, camera, and VoLTE. Table 2 shows the comparison of Android OS with other OS where various features of the different smart-phone OS are summarized. This section explores Android OS architecture, its security model, and security issues.

2.1 | Android architecture

As shown in Figure 4, Android architecture is distributed into four layers and described as follows:

Layer 1-Linux kernel: This layer (or bottom layer) is a modified Linux kernel that consists of hardware drivers such as camera, Blue-tooth, universal serial bus, Wi-Fi, audio, display, and so forth. Besides drivers, it also provides basic functionalities of memory, process, and power management to make sure that each running app should get the required memory, run in a process, and react to different power modes, respectively. This layer also provides an interface for hardware abstraction to implement functionality without modifying the higher-level system. Earlier Android developers change large segments of the Android source code to update the device, but later smart-phone devices with Android version 8.0 or higher provides an interface for manufacturers to update the Android devices to a newer version easily by replacing the Android framework without rebuilding the hardware abstraction.[20]

Layer 2-Libraries and Android RunTime: This layer (runs on top of Linux kernel) contains libraries written in C/C++ languages. These Android libraries include surface manager (to compose windows on the screen), media framework (to record audio and video), SQLite (for storage and sharing of application data), open graphics library for embedded system (to provide Java interface for OpenGL three-dimensional [3D] graphics), FreeType (for font rendering), WebKit (to display Internet content), scalable graphics library (two-dimensional [2D] graphics to draw application program interface), secure socket layer (for Internet security), and libc (System C library). This layer contains Android runtime (ART) for Dalvik virtual machine (DVM) and core libraries. The DVM for Android OS works exactly like Java virtual machine (JVM). The DVM is responsible to convert byte-code generated by the Java compiler to Dalvik Executables (dex) files. Earlier all Java written apps run separately within their DVM, but later smart-phone devices with Android version 5.0 or higher run each app in its process and with its instance of the ART. This is done so that multiple virtual machines can run on low-memory devices by executing dex files.[21]

Layer 3-Application framework: This layer provides a set of services to Android apps which are used by application developers in layer 4. This layer manages basic functions of Android devices and provides services such as activity manager (to manage the lifecycle activity of apps), window

TABLE 2 Smart-phone operating system (OS) comparison

Reference	Operating system	Security	Kernel	Supported platforms	Programming languages	Packaging	OS code	Smart-phone examples
15	Android	Dalvik VM	Linux	X86, ARM, MIPS	Java	Android package	Open	Google Pixel Samsung Galaxy
16	iOS	Common data security architecture	Hybrid (Darwin/XNU)	ARM	Objective-C	iPhone application archive	Closed	iPhone X_S, iPhone X_R
17	Windows	Signed applications	Windows CE	ARM	C# and VB	Windows phone package	Closed	Microsoft Lumia
18	Symbian	Cryptographic, hashing and random number generating algorithm	Microkernel (EKA2)	X86, ARM	Java ME, C++	Software installation script, Java application descriptor	Closed	Nokia N95
19	BlackBerry	Platform and application services	Java based	ARM	Java	JAR	Closed	BlackBerry Evolve

Abbreviations: ARM, advanced RISC machine; MIPS, microprocessor without interlocked pipeline stages.

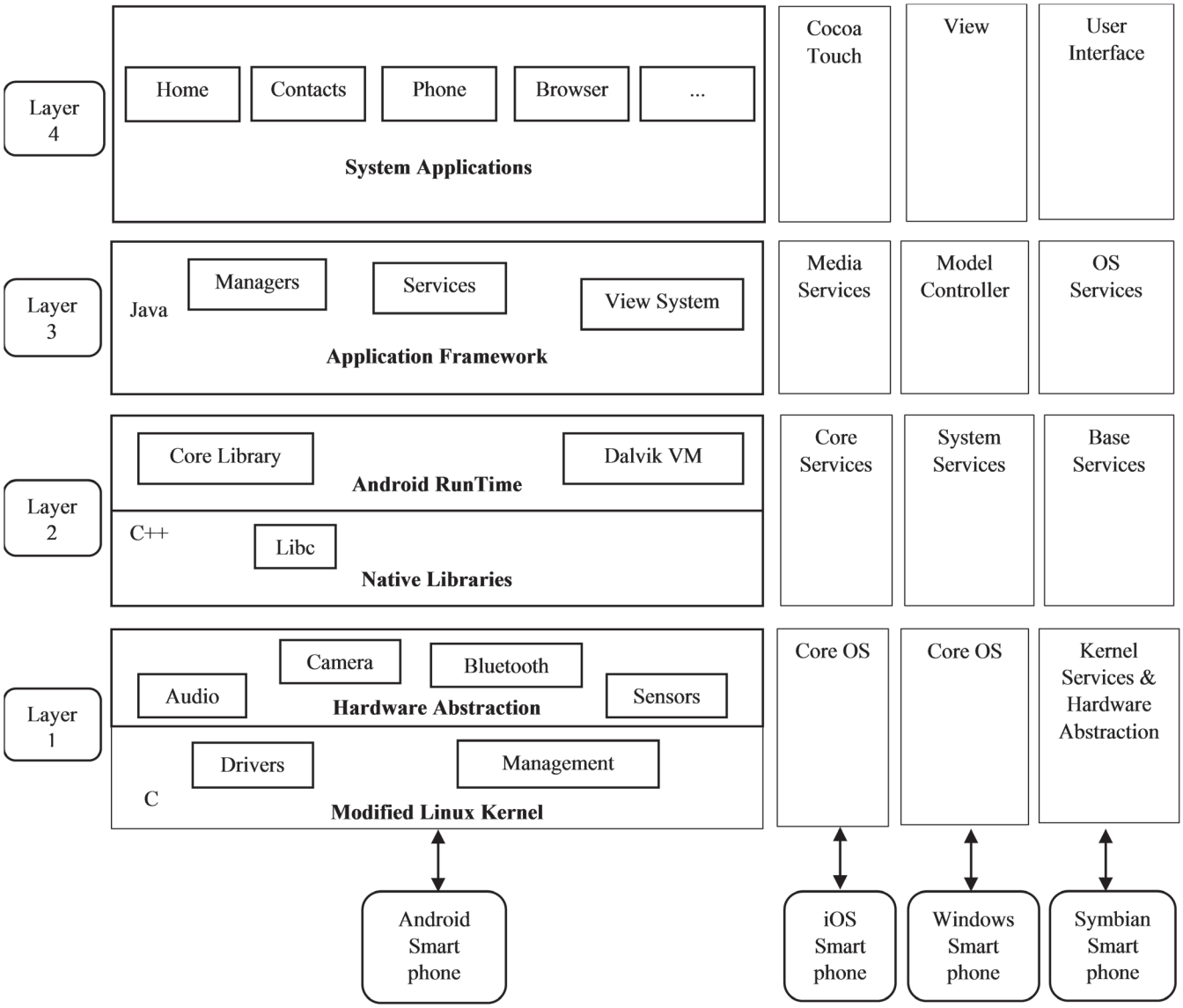

FIGURE 4 Android, iOS, Windows, Symbian operating system architecture

manager (to manage the layout and visibility of Android windows), content provider (to provide an interface for sharing data among apps), view system (to build user interfaces of apps), notification manager (to display alerts), telephony manager (to manage voice calls), resource manager (to access graphics, strings, layout files, and color settings), location manager (to manage the locations), and extensible messaging and presence protocol service (for instant messaging).

Layer 4-Applications: This layer (or top-most layer) allows developers and users to install the apps and interact with the device. It contains native apps (web browser, calendars, contacts, and so forth) and third-party apps installed by the users (games, dictionary, and so forth).

2.2 | Android security model

Android OS takes advantage of the Linux Kernel (Layer 1) to provide security by isolating the resources from each other. The Android Linux assigns User ID (UID) to apps and uses app sandboxing for isolation of apps. The security model of Android OS is described as follows:

App sandboxing: The main principle of employing app sandboxing in Android devices is to isolate the simultaneously running apps (on the same device) from each other to prevent an app from accessing data of another app without authorization. To fulfill this goal, the UID (or App-ID) is allotted to an installed app for execution in an isolated process where each app has a dedicated data directory for which only it has permission to read and write to Reference 22. In this way, the apps are isolated from each other at the process as well as file level. The app sandboxing provides security

to Android OS by preventing the interaction of genuine apps with malicious apps. However, in Android version 5.0 or higher, developers introduced Security-Enhanced Linux (SELinux) to enforce mandatory access control separation between the device and apps.

Permissions: As app sandboxing restricts apps from sharing resources with other apps. Thus, granting resources to the apps is one of the major goals in the Android security model. To achieve this, Android developers used permissions for apps to share resources such as hardware, Wi-Fi, camera, and so forth. The Android permissions provide security to users by supporting their privacy after informing them about permissions obtained by an app. For example, if an app asks to access the camera, gallery, location, and contacts; then a pop-up will be displayed on the device which will ask users to either accept or deny the access. Android permissions are bifurcated into 4 levels,[23] namely, Normal, Dangerous, Signature, and SignatureOrSystem as shown in Table 3. Out of these, Dangerous permissions possess a high level of risk because it asks for sensitive data, so the user confirmation to grant permission is compulsory during the installation of the app.

Android app developers define custom permissions in <permission> element of the AndroidManifest.xml file for managing the access to components in such a way that the predefined device permissions will not be used.[24] Therefore, multiple apps might use the same custom permissions by describing them in the <permission> element of the AndroidManifest.xml file. For example, if two apps provide the same type of capabilities, they might get the same logical name for their custom permissions. Thus, the permission model has been dramatically changed after a major update with Android version 5.0 or higher. The update possess a constraint on custom permissions for apps that are signed with different keys such that only a single app on a smart-phone device can define given custom permission unless another app that defines the permission is signed with the same key. Thus, if a user wants to install an app with identical custom permissions which are already being used by another app on the same device and the app is not signed with the same key as the resident app; the device immediately blocks the installation of the app.

Encryption: Encryption is a cryptography technique to encrypt data into a ciphertext with the help of encryption keys.[25] Android OS developers provide a full file-system encryption feature for users' privacy such that all data in the device can be encrypted in the Linux kernel. The principle behind performing data encryption is to protect all users' data; when the OS suspects that the phone might be out of the user's control. For example, when the users' Android-based smart-phone device is switched-off; the data will be decrypted when the user restarts the smart-phone device and unlocks the screen with a PIN or defined pattern. This often causes a small delay while rebooting the smart-phone device. But this feature didn't support full-disk encryption to protect the user-data partition of the smart-phone device. However, in Android version 5.0 or higher, developers introduced full-disk encryption with a single key which is protected by the device password itself. Later on, in Android version 7.0 or higher, developers introduced file-based encryption that allows different files to be encrypted with different keys which can be unlocked independently.[26]

Play Store: Play Store (or Android Market) is an online store where users can download apps developed by Google or third-party app developers. Apart from it, users can also download apps from unknown sources by enabling the following path: *Settings-> Additional Settings-> Privacy-> Unknown Sources*

Since Android has an open platform, thus, anybody can write and deploy the apps which may lead users to download and install malicious apps. Thus, in Android version 7.0 or later, Google developed Play Protect which continuously scans the smart-phone device to detect and remove malicious apps.[31] Another security service implemented by Google is known as Bouncer, which is a dynamic analysis sand-boxed environment to protect the Play Store from malicious apps.

2.3 | Security issues in Android OS

Though Android OS has a robust security model but there are some loopholes that cyber-criminals can exploit to compromise the security of Android phones. These security issues are discussed as follows:

Android App Market: Android users can install malicious apps from the app market because of limited mobile app vetting process[3] to assure mobile apps comply with the security requirements of the agency. There are a lot of malicious apps found in Google Play Store such as Fast Charging app,[32] Fast Charger and Battery Saver app,[33] and SMS Blocker app.[34] Moreover, users can install malicious apps from nonmarket sources which further gives entry to ransomware.

TABLE 3 Android permissions

Reference	Permission value	Risk level	Grant	Access	Example
27	Normal	Low	Automatically	Isolated apps	WAKE_LOCK
28	Dangerous	High	User confirmation required	Private data	WRITE_EXTERNAL_STORAGE
29	Signature	Low	Developer certificate required	Resources	BIND_DEVICE_ADMIN
30	SignatureOrSystem	Low	Either preinstalled in Android operating system or developer certificate required	Resources	BIND_WALLPAPER

Permissions: Overclaiming of permissions is a most drastic issue in Android OS which breaks the principle of least privilege and leads to information and monetary loss.[35] If a Dictionary app requests unnecessary permission such as READ_PHONE_STATE; the usage of permission can be exploited by sending phone state (IMEI number) to the attacker without users' consent. In Reference 36, a scenario of exploiting permissions has been presented by finding a flaw in the Android Permission scheme by creating a malicious app that accesses the resources of another app by setting the same permission name. There are various Android apps present in Play Store which ask for unnecessary permissions from users such as a Battery Saver app[33] asks for Camera access and requires personal information (including photos and files).

Native Code: Native code written in C/C++ execute in libraries outside the Dalvik VM and the code is sandboxed by combining user-id and group-id. The native code is more flexible as it operates near to hardware and can manipulate the memory.[37] Moreover, the OS kernel system call interface is directly exposed to the native code. Thus, all local root exploits by malicious apps are employed as native code or in binary form.

Discussion: This section presented an overview of the Android OS by explaining its architecture and security model. The security issues present in the Android OS can be exploited by cyber-criminals to threaten the privacy of users. The following section presents the threat scenario of ransomware on Android OS.

3 | ANDROID RANSOMWARE

Ransomware is a kind of malware, but as the name suggests, it demands ransom from the victims to liberate the seized material.[38] Ransomware is not a PC problem any-more; other devices such as mobile phones, televisions, and Internet of Things (IoT) devices are also vulnerable.[39] Ransomware can be of two types: Lock-screen and Crypto ransomware. In the former, ransomware hijacks the device by locking the screen while in the latter, it encrypts the users' documents (including photos and files). Besides this, the Android ransomware app masquerades as a benign app by taking the same name and icon to confuse the users. Examples of Android ransomware include Android/Simplocker and Android/Lockerpin. Table 4demonstrates the time-line of Android ransomware families from their emergence (2013) till date (2020).

The complete threat scenario of Android ransomware is shown in Figure 5 and discussed as follows:

- An attacker inserts malicious code in a benign app which requires installation on the users' device to perform harmful actions. Various apps related to games or pornography are often chosen by attackers to increase the infection vector. It mostly targets apps that are most likely downloaded by teenagers as they can't discriminate malicious and benign apps. For example, Gaming apps (car and bike racing), Flash Player, and so forth. The Android ransomware can disguise as a fake antivirus or fake banking app (details are given in Table 4).
- During installation, it asks users to grant permissions such as WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE, BIND_DEVICE_ADMIN, KILL_BACKGROUND_PROCESSES, and more. If a user grants these permissions, then successful installation of ransomware is marked.
- After getting installed, it starts its noxious operation to harm the Android device. It scans users' documents with extensions—JPEG, TXT, and DOV in the memory card or internal memory of the phone.
- After scanning, it establishes a communication channel with Command and Control (C&C) server to receive commands from attacker through Tor network or HTTP protocol. Then, it executes the commands sent by the malware writer(s) and transfers confidential information of users such as photos, documents, contacts, device model, IMEI number to the C&C server.
- After acquiring the Android-based smart-phone, ransomware kills processes of antimalware apps running in the background with KILL_BACK-GROUND_PROCESSES permission to evade itself from being detected by antimalware.
- It can either lock the device, encrypt documents (with Advanced Encryption Standard algorithm), or both after following commands from attackers. This produces a botnet of infected Android-based smart-phone devices under the attacker's control. The Andro/LockerPin[50] ransomware uses administrative permissions to set or change the lock screen PIN. It also changes the wallpaper of the phone to display a scary message from the FBI on the screen to threaten the victim to pay the fine for watching illegal content. It may also click the victim's photo with a front camera and display it along with the message.
- It demands ransom in the form of crypto-currency (such as Bitcoin) for fast transactions. Even if the victim pays the ransom, it is not assured that the ransomware will unlock the phone and decrypt the documents. Afterward, it increases its infection vector by sending a malicious link to the victim's contacts in the form of SMS.

Discussion: This section presented a threat scenario of ransomware on Android-based smart-phones. The following section surveys and examines the analysis methods for feature extraction of Android ransomware which is a preprocessing step for machine learning techniques.

TABLE 4 Android ransomware

Android ransomware	Category	Year	Type	Consequences	Infection vectors	Encryption algorithm	Permissions	Example
Android Defender	Fake antivirus	June 2013	Locker	Device crash, Monetary loss, Credential loss	Third party app (Rogue Antivirus)	X	OS settings, Display pop-up window	Android/FakeAV.E[8]
Android Police	Fake Law agency	May 2014	Locker	Monetary loss	Adult Video	X	Set wallpaper	Android/Koler and Android/Locker[8]
Simplocker	Fake application	May 2014	Locker and Crypto	Encrypt files, Monetary loss	Embedded in an app (Third party)	AES	Camera, Internet, READ/WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE, ACCESS_NETWORK_STATE	Android/Simplocker.I[8]
Pletor	Fake application	June 2014	Locker and Crypto	Encrypt files, Monetary loss	Adult websites	AES	Contacts, Camera, Calls, Network connection, system settings, TYPE_SYSTEM_ALERT, system logs	Ransom.AndroidOS. Pletor.a[40]
Jisut	Prank	June 2014	Locker	Monetary loss	SMS	X	Administrative access	Android/LockScreen. Jisut[8]
Small	Fake Law agency	June 2014	Locker and Crypto	Encrypt files, Steal information, Monetary loss	SMS	?	Administrative access	TrojanRansom.AndroidOS. Small[41]
Svpeng	Fake Law agency	June 2014	Locker	Call list, Browser history, Camera, Monetary Loss	Adult websites	X	Administrative access	Banker.AndroidOS. Svpeng[42]
LockDroid	Fake Law agency	October 2014	Locker	Browser history, SMS, call records, Monetary loss	App update package	X	Administrative access	Android.Lockdroid.E[43]
Fusob	Fake Law agency	January 2015	Locker and Crypto	Steal device information, Camera, Location, Contacts, Monetary Loss	Third party app	AES	Administrative access	Trojan- Ransom.AndroidOS. Fusob[41]
Xbot	Fake banking app	May 2015	Locker and Crypto	Steal banking credentials, SMS, Contacts, Monetary Loss	Phishing	XOR each byte with fixed integer	Administrative access	hxxp://market155[.]ru/Install.apk[44]
LockerPin	Fake Law agency	August 2015	Locker	Data loss, Monetary loss	Embedded in an app (Third party)	X	Administrative access	Android/LockerPin[8]
Charger	Fake app on Play Store	January 2017	Locker	Data theft, Location access, Monetary loss	Embedded in an app (Play Store)	X	Administrative access	EnergyRescue[45]
WannaLocker	Plugin	June 2017	Locker and Crypto	Data theft, Monetary loss	Plugin for gaming app	AES	Set wallpaper, Modify audio settings, ACCESS_NETWORK_STATE, READ/WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE	Android. WannaLocker[46]
DoubleLocker	Fake banking app	October 2017	Locker and Crypto	Wipe bank account, Data theft, Monetary loss	Adobe Flash Player	AES	Accessibility service, Administrative access	Android. DoubleLocker[47]
CovidLock	Fake COVID19 Tracker app	March 2020	Locker	Steal credentials of social media accounts, Data theft, Monetary loss	SMS, e-mail	X	Accessibility service, Administrative access	Android. CovidLock[48]
Dubbed Black Rose Lucy	Fake FBI app	April 2020	Crypto	Steal credit card information, Data theft, Monetary loss	social media, messenger	AES	Accessibility service, Administrative access	Android.Lucy[49]

Abbreviations: X, not applicable, ?, information not found.

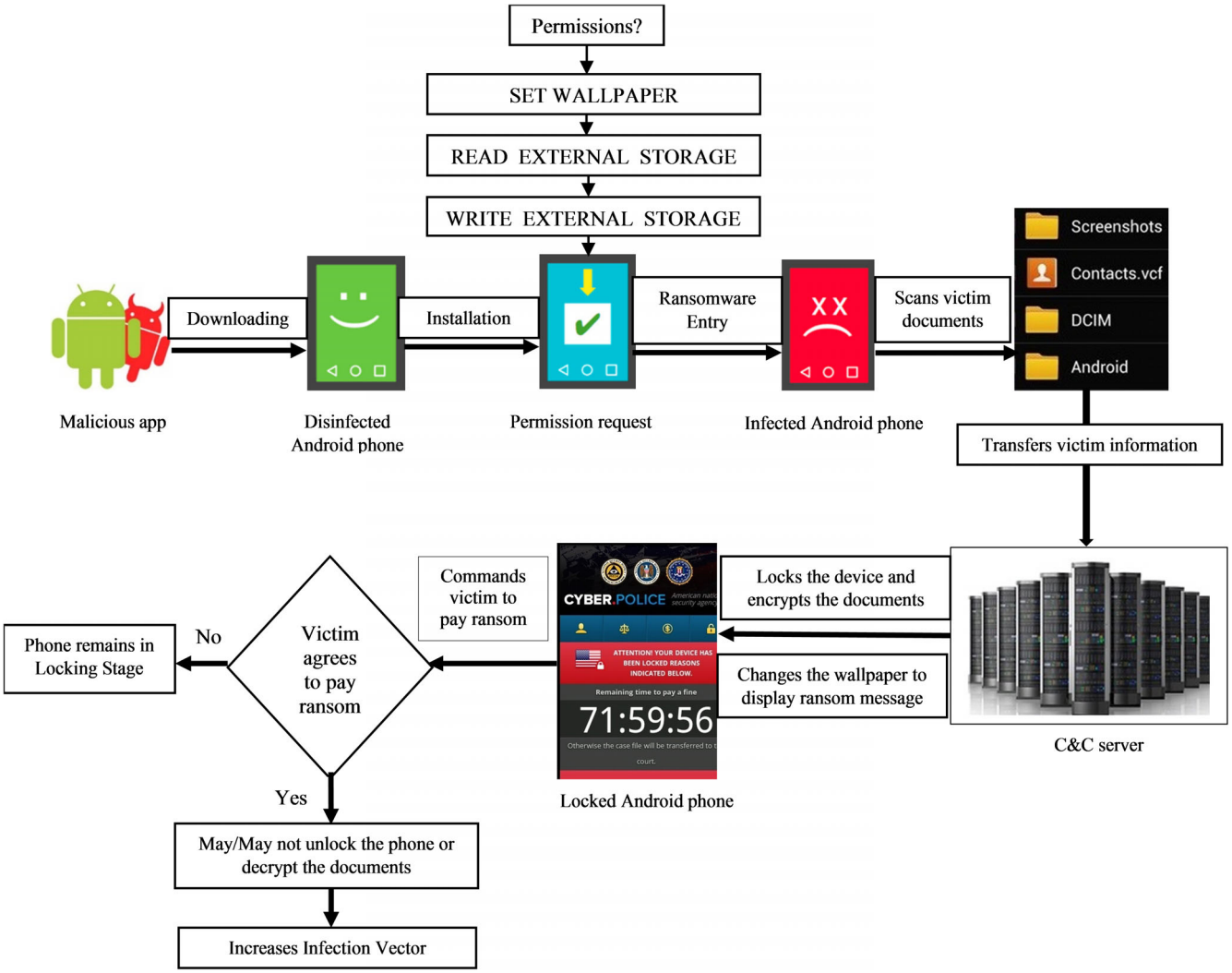

FIGURE 5 Threat scenario of Android ransomware

4 | ANDROID RANSOMWARE: ANALYSIS METHODS

The machine learning techniques cannot work without the datasets; thus before surveying analysis methods for feature extraction; the available datasets of Android ransomware are needed to be explored. The Android ransomware datasets have been provided by HelDroid[51] and RansomProber.[52] The HelDroid dataset collected by the researchers in the year 2015 contains 670 Android package kits (APKs) of Android ransomware. The RansomProber dataset collected by the researchers in the year 2017 contains 2300 APKs of Android ransomware. The data is available for other researchers and practitioners to replicate the study and analyze the techniques. The available Android malware datasets including Android ransomware are examined by Sharma et al.[53] where they found that Android ransomware APKs in RansomProber dataset are one of the least detected apps by antimalware software.

Thus, there is a requirement of robust methods to perform in-depth analysis to extract features of Android ransomware. In this section, we survey static, dynamic, and hybrid analysis methods:

4.1 | Static analysis

Static analysis implies extracting the features of Android ransomware without executing the APKs in an Android emulator. The APKs are compressed files created by the Android OS developers for transmitting and installing apps in Android-based smart-phones. The reverse engineering tools (apktool, dex2jar, ded, dare, and so forth) are used to disassemble APKs to read the source code written in various formats such as XML, dex, JAVA, smali, and so forth. Then, static analysis methods are used to analyze these code segments for feature extraction. The following files are statically analyzed by researchers to extract features of Android ransomware:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
.............
.............
<action android:name="android.intent.action.BOOT_COMPLETED"/>
<action android:name="android.intent.action.SCREEN_ON"/>
<action android:name="android.app.action.DEVICE_ADMIN_ENABLED"/>
```

FIGURE 6 Permissions and Intents requested by Android Simplocker ransomware in AndroidManifest.xml file

4.1.1 | AndroidManifest.xml file

The permissions and intents listed in the AndroidManifest.xml file are important features to classify malicious and genuine apps. Ransomware can request a large number of permissions and access dangerous permissions from users as compared with benign apps. For example, as shown in Figure 6, Simplocker ransomware* requests for dangerous permissions such as READ_CONTACTS, READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, SEND_SMS, and RECEIVE_SMS which are commonly asked by malicious apps but rarely asked by nonmalicious apps. Similarly, the AndroidManifest.xml file contains intents which are abstract objects used by an app to switch users to another app based on actions it would like to perform (e.g., showing a map, sending an SMS, clicking a photo). Ransomware can use intents to navigate among activities to perform malicious actions. For example, as shown in Figure 6, Simplocker ransomware† waits for SCREEN_ON to start its malign activities when the screen of the device becomes interactive.

Applications

Ransomware inside out:[54] The authors collected Android ransomware samples from the HelDroid dataset to perform static analysis on the AndroidManifest.xml file. They analyzed that the Simplocker ransomware sends users' personal information from a phone to a remote server after acquiring READ_PHONE_STATE permission. The Android ransomware misuses READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE permissions to perform encryption operations on the external storage of the device. Besides this, DeviceAdminReceiver class of Android ransomware waits for intent (DEVICE_ADMIN_ENABLED) to get administrative privileges.

Effectiveness of permissions:[55] The authors collected Android ransomware samples from HelDroid, RansomProber, VirusTotal, and Koodous dataset to perform static analysis on the AndroidManifest.xml file. They analyzed the common and distinguished permissions asked by ransomware and benign apps. They found that approx. 90% of Android ransomware samples in these datasets requested for RECEIVE_BOOT_COMPLETED permission and 99.5% benign samples requested for INTERNET permission.

RansomAnalysis:[56] The authors collected Android ransomware samples from the RansomProber dataset to perform static analysis on the AndroidManifest.xml file. They analyzed the top 10 permissions requested by Android ransomware where they found that RECEIVE_BOOT_COMPLETED permission has been used by all Android ransomware samples to check the booting status of the smart-phone.

4.1.2 | Classes.dex file

In Android OS, programs are written in JAVA language are first compiled into .dex format which further zipped into an application package (.apk) file with the aapt tool. The research community developed reverse engineering tools (such as ApkTool, Dedexer, Dex2jar, Ded, and Dare) to convert dex files into Dalvik and JAVA bytecode for easy reading and manipulation. Ransomware can use Dalvik or Java bytecode to declare methods to perform locking and/or encryption operations on the users' phone. For example, *Cipher.doFinal* method contains malicious code to encrypt users documents.

*MD5: 00fa6fb96ce806d0064b3e426038dcef

Applications

R-PackDroid:[57] The authors collected Android ransomware samples from HelDroid and VirusTotal datasets to perform static analysis on dex files. They used the ApkTool to convert dex files into Dalvik bytecode to analyze invoke-type instructions. They analyzed that javax/crypto and java/security instructions present in the crypto-related application programming interface (API) package list were used by Android ransomware samples to perform malicious activities.

Ransomware steals your phone:[58] The authors collected Android ransomware samples from the HelDroid dataset to perform static analysis on Java Bytecode. They used the dex2jar tool to convert dex files into JAVA bytecode. They applied a transform operator on Java bytecode for parsing and transforming it into the calculus of communicating systems. They also used mu-calculus logic to present the behavior of Android ransomware and applied the concurrency workbench of new century model checker to detect Android ransomware.

Structural entropy-based classification:[59] The authors collected Android ransomware samples from the VirusTotal dataset to perform static analysis on dex files. They calculated the similarity score of the ransomware and benign samples with structural entropy by using discrete wavelet transform to segment dex files according to different entropy levels. They calculated the distance between each sequence segment to find the similarity among dex files of ransomware and benign samples.

4.2 | Dynamic analysis

In contrast to static analysis, this method extracts features by executing the APKs in an Android emulator. The dynamic analysis methods deal with features that involve code and system calls loading during the execution of an app at run-time. Dynamic analysis can be performed by running an Android app in user/kernel space, emulator, and virtual machine (which are discussed in Reference 60). The following files are dynamically analyzed by researchers to extract dynamic features of Android ransomware:

4.2.1 | Process monitoring

In Android OS, each application runs in its process and instance of the Android ART. The App-ID in Android is allotted to an app during installation so that it will be executed in an isolated process. However, ransomware can access these processes to run its malicious code with genuine processes. For example, ransomware can create a malicious process to copy and access files from the storage.

Application

Ransomware prevention technique:[61] The authors performed the dynamic analysis to monitor processors, memory, and input-output usages consumed by each process. They analyzed that the infected processes consumed more system resources as compared with genuine processes. The malicious processes performed encryption operations and transferred documents to a remote location. After detection of malicious processes accessed by Android ransomware, the authors stopped working on those processes and deleted the programs associated with them.

4.2.2 | User interaction

The Android apps notify users before doing any sensitive activity through the user interface. But the ransomware can misuse this functionality via malicious apps so that the users allow the rights to perform malicious actions. For example, malicious behavior such as fake phone calls and malicious SMS were initiated by cyber-criminals to trap users by installing fake apps.

Application

RansomProber:[52] The authors created an Android ransomware dataset known as the RansomProber dataset. They checked whether the encryption process was launched by an app or initiated by users. For this, they performed dynamic analysis on widgets (such as labels and list-views) and activities (such as capturing photos and dialing phone). The authors recorded all coordinates of users' click on layout. During the observation, if they didn't find any click operation for file encryption initiated by users. Then it was considered as an abnormal activity done by Android ransomware.

4.3 | Hybrid analysis

The static analysis cannot deal with code obfuscation and run-time activities whereas the dynamic analysis suffers from code coverage issues. Thus, the hybrid analysis methods can be used to merge the advantages of static and dynamic analysis methods. This method can be achieved by using both static and dynamic analysis methods for feature extraction. The hybrid analysis increases robustness, code coverage, finds more features, and monitors edited apps as compared with other analysis methods.[5] The following files are statically and dynamically analyzed by researchers to extract features of Android ransomware:

4.3.1 | Resource and Smali files with network traffic

The Android apps use resource files (e.g., layout, values) to display static content on users' devices such as text, colors, images, and so forth. But ransomware can misuse the resource files to display threatening text on users' screen. The threatening text can be in any form such as quotes, payment information, law-breaking messages, and so forth. Figure 7 shows a layout file where sentences written in double-quotes mark the threatening texts displayed by Android ransomware. Besides this, smali is an assembly language that runs on Android DVM and can be obtained by decompiling dex files with the ApkTool.[62] Every class in the Java code has a separate smali file that includes instructions, methods, and function calls. But the ransomware can misuse smali files to create malicious locking and encryption operations. On the other side, the Android apps need network connections for communication to the remote server, where a ransomware sample misuse this functionality to send users' personal information to a remote server.[63]

Application

HelDroid:[51] The authors created an Android ransomware dataset known as the HelDroid dataset. They performed static analysis on smali code to search for lockNow(), onKeyDown(), and onKeyUp() methods for detection of screen-locking. They also performed static taint analysis on Smali code to observe encrypting operations such as CipherOutputStream(), Environment.getExternalStorageDi-rectory(), and so forth. Besides this, they performed static analysis on text present in the resource files and dynamic analysis on network traffic. They executed ransomware samples in a virtual environment to capture network traffic and extracted dynamically allocated text from it. They used natural language processing technique to detect threatening messages displayed by Android ransomware on users' mobile screen. They classified the sentences as scary, payment, porn, law, and copyright to detect Android ransomware.

4.3.2 | AndroidManifest.xml and Smali file with directories

The AndroidManifest.xml and smali files are already discussed in Sections 4.1.1 and 4.3.1, respectively. The system paths are the paths to the Android directories which contain users' documents and apps information. But the ransomware can access these directories after getting administrative privileges and send users' personal information to the attacker at a remote location.

Application

Automated detection and analysis for Android ransomware:[64] The authors introduced an idea for the implementation of hybrid analysis where the static analysis can be performed on permissions and API call sequences. They observed that ransomware samples used a similar kind of sequence

```
<TextView android:text="YOUR DEVICE HAS BEEN BLOCKED"/>

<TextView android:text="DETECTED ILLEGAL CONTENT"/>

<TextView android:text="You have been subjected to violation of Copyright and Related Rights
Law (Video,Music,Software) and illegally using or distributing copyrighted contents.
Article 1,Section 8,Clause 8 of the Criminal Code provides for a fine of two to five hundred
minimal wages or a deprivation of liberty for two to eight years." />

<TextView android:text="You have also been viewing or distributing prohibited Pornographic
content. Article 202 of the Criminal Code provides for a deprivation of liberty for four
to twelve years."/>

<TextView android:text="Pursuant to the amendment to Criminal Code of United States of
America of May 28, 2011, this law infringement (if it is not repeated - first time) may be
considered as conditional in case you pay fine of the States."/>

<TextView android:text="To unlock your device and to avoid other legal consequences, you are
obligated to pay a release fee of $300. Payable through GreenDot MoneyPak (you have to
purchase MoneyPak card. load it with $300 and enter the code). You can buy the card at any
store or gas station, payzone or paypoint."/>
```

FIGURE 7 Threatening messages displayed by Android ransomware

TABLE 5 Analysis methods for feature extraction of Android ransomware

Methods \ Applications	RIO[54]	RSP[58]	EoP[55]	RansomAnalysis[56]	RPT[61]	R-PackDroid[57]	RansomProber[52]	HelDroid[51]	ADAAR[64]	RanDroid[65]	RTARD[66]
Static analysis											
Permissions	●	○	●	●	○	○	○	○	●	○	○
Intents	●	○	○	●	○	○	○	○	○	○	○
Java bytecode	●	●	○	○	○	○	○	○	○	○	○
Dalvik bytecode	○	○	○	○	○	●	○	○	○	○	○
Locking methods	○	○	○	○	○	○	○	●	○	○	○
Encryption methods	○	○	○	○	○	○	○	●	○	○	●
Threatening text	○	○	○	○	○	○	●	○	●	○	○
Application Program Interface	○	○	○	○	○	○	○	○	●	○	●
Dynamic analysis											
Process monitoring	○	○	○	○	●	○	○	○	○	○	○
User interaction	○	○	○	○	○	○	●	○	○	●	○
Network traffic	○	○	○	○	○	○	○	●	○	○	○
Directories	○	○	○	○	○	○	○	○	●	○	○
Malicious domain names	○	○	○	○	○	○	○	○	●	○	○
System calls	○	○	○	○	○	○	○	○	○	○	●

Abbreviations: ADAAR, automated detection and analysis for Android ransomware; EoP, effectiveness of permissions; RIO, ransomware inside out; RPT, ransomware prevention technique; RSP, ransomware steals your phone.

while calling the methods. The dynamic analysis can be performed to check the system paths (e.g., /data/system/accounts.db) for Android directories. The authors analyzed that Android ransomware creates files (creatSDFile), directory (creatSDDir), and writes data (write2SDF) in secure digital card. They also analyzed that dynamic analysis can be performed on the malicious domain; where ransomware sends users' personal information and put malign charges to give the data back.

Discussion: This section described available Android ransomware datasets which can be analyzed for feature extraction. Various types of analysis methods were explained with different features extracted by researchers in the literature. The summarized representation of these analysis techniques with applications is shown in Table 5. The following section describes machine learning techniques for the detection of Android ransomware.

5 | ANDROID RANSOMWARE: DETECTION METHODS

The features extracted after applying various types of analysis methods will be used by machine learning techniques to diagnose the malign behavior of Android ransomware. Machine Learning is an area of artificial intelligence that gives computers the ability to learn without being explicitly programmed.[67] It can detect malicious patterns and predict future behavior to stay one step ahead of cyber-criminals. It is used to predict unknown malicious behavior with the help of existing knowledge. Various machine learning algorithms employed by researchers for the detection of Android ransomware are discussed in the following:

5.1 | Hidden Markov models

Summary: A hidden Markov model (HMM) is a statistical tool that belongs to the category of Markov Models. Markov chain is a weighted automaton that consists of states connected through transition probabilities. Consider the following simple state transition: state 1->state 2 which are interconnected to each other so that state 1 can be reachable from state 2 and vice-versa. An HMM permits both observed and hidden states in the probabilistic model where the observed states are visible in the input while hidden states are the causal factors that are partially observable.[68] Besides states and transition probabilities, an HMM consists of a set of output symbols and output probabilities. The actual states in HMM are

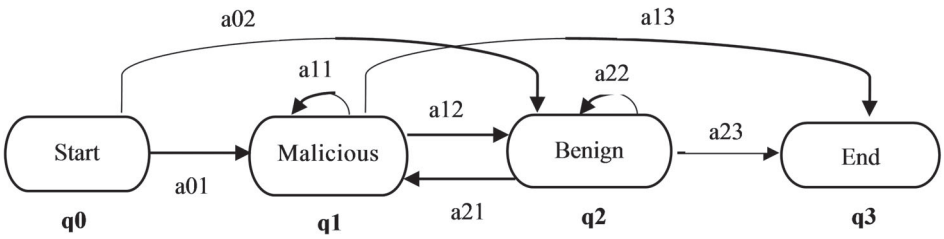

FIGURE 8 Hidden Markov model example

hidden but the output produced by the model is observable. Figure 8 shows an HMM example for the detection of malicious apps. The components used by an HMM are *Q* (set of N states), *A* (state transition probability matrix), *O* (observations), Π (initial probability distribution over states). The distinct states (Q) of the Markov process are- malicious and benign. The observations are known data such as screen lock, encryption, sending SMS to premium numbers, accessing location, and camera.

Android ransomware detection: The authors in Reference 69 performed static analysis and employed the HMM model for detection of Android ransomware. They collected benign samples from Play Store and Android ransomware samples from the HelDroid dataset. They disassembled dex files into smali codes to perform static analysis on opcodes sequence. The authors collected all invoke-instructions in a single file to create a sequence of all instructions. They applied the HMM model with *N* hidden states ($N = 3, 4, 5$) and used the Baum–Welch algorithm in each hidden state to calculate the values of unknown parameters. They compared the obtained opcode sequence of ransomware apps with genuine apps for classification.

5.2 | *K*-nearest neighbors

Summary: The *K*-nearest neighbors (*k*NN) is an instance-based lazy learning classification algorithm that has very less or no prior knowledge about data distribution. In *k*NN, the similarity of two objects in a single class is calculated by a distance metric.[70] The classification of an unknown class is done in the same category as its first nearest neighbor or determined by the votes of *K*-nearest neighbors (*k* is a positive integer). The value of *k* and nearest neighbors are calculated by empirical cross-validation and distance measure, respectively.

Android ransomware detection: The authors in Reference 71 performed hybrid analysis and used *k*NN classifier for detection of Android ransomware, spyware, Trojans, and botnet. They performed static analysis on dangerous permissions and dynamic analysis on system calls and user interactions to extract features of Android ransomware. Besides this, they monitored the installation of new apps, apps producing a large number of processes, user activities, and apps running in the foreground to detect spyware and Botnet. They also monitored the SMS and contact list to detect SMS Trojans. The authors used *k*NN classifier, where the Euclidean distance was used to calculate the similarity of apps. The *k*NN classifier with $k = 1$ was trained with genuine as well as malicious apps to allocate it to the nearest neighbor with the highest similarity.

The authors in Reference 72 performed static analysis and employed *k*NN classifier for detection of Android ransomware. They performed static analysis on native code to extract features of Android ransomware. After feature extraction, they applied principal component analysis technique for dimensionality reduction of features. They applied *k*NN, random forest (RF), support vector machine (SVM), Naive Bayes (NB), and artificial neural networks (ANNs) to detect Android ransomware. The results showed that *k*NN and RF gave the best results as compared with other machine learning algorithms for the detection of Android ransomware.

5.3 | Decision trees

Summary: A decision tree (DT) is a rule-based tree to split the dataset into different classes. It uses a top-down approach to split the heterogeneous data for the classification of malicious and benign apps. A DT uses the C4.5 algorithm to select the best attribute to create a node based on entropy values.[73] The data present at the root node of the tree has the highest entropy, and therefore, it is broken down into smaller subsets to reduce the value of entropy. This process is repeated recursively till further classification is impractical or the leaf node has an entropy value equals to zero. A lower value of entropy implies higher information gain (IG), and the attribute which provides the maximum IG, is chosen as the best path in DT. Entropy and IG are calculated as follows:

$$\text{Entropy}(T) = \Sigma_{i=1}^{T} - p_i \times \log_2(p_i), \tag{1}$$

$$\text{Entropy}(T, S) = \Sigma_{i=1}^{c} P(c) \times \text{Entropy}(c), \tag{2}$$

$$\mathrm{IG}(T, S) = \mathrm{Entropy}(T) - \mathrm{Entropy}(T, S), \tag{3}$$

where,

- Entropy (T) is the value of entropy calculated before the split.
- p_i is the probability of apps downloaded from Play Store divided by the total number of apps.
- Entropy (T, S) is the value of entropy calculated after the split.
- Entropy (c) is the computed value of entropy of left and right split.
- $P(c)$ is the probability of the number of malicious apps (on the left branch) and benign apps (on the right branch) divided by the total number of apps, respectively.
- IG is the information gain which is calculated by comparing entropy before and after the split.

As shown in Figure 9, apps can either be downloaded through Play Store or unknown sources. At the root node, the value of entropy is high, and it is unable to judge whether it is a malicious or benign app. Thus, the data is again split based on permissions acquired by Android apps to reduce the entropy. If the downloaded app asks for dangerous permissions (e.g., SEND_SMS, READ_CONTACTS) then, it is classified as a malicious app.

The RF classifier combines the functionalities of DTs to create the ensemble learning algorithms. Unlike DTs, it randomly chooses data features (attributes) as inputs. For example, if the total number of features is N, then RF samples $\sqrt{N}$ subfeatures. As the number of trees in the forest increases, the variance of the model decreases, and bias can be removed by taking an average of all trees.

Android ransomware detection: The authors in Reference 74 performed hybrid analysis and employed the DT (J48) algorithm for the detection of Android ransomware. The J48 classifier is an application of the C4.5 algorithm to generate pruned or unpruned DTs. It decides the value of the new sample with the help of values given in the available data. The authors collected benign samples from Play Store and Android ransomware samples from the HelDroid dataset. They disassembled dex files into smali files to perform static analysis on n-grams opcodes ($n = 2$). They executed each sample in an Android emulator to extract dynamic features such as CPU usage, network traffic logs, system calls, and occupied memory. They compared DT (J48) with NB and logistic regression (LR) algorithms where DT gave the best results for detection of Android ransomware.

The authors in Reference 75 performed static analysis and employed an RF algorithm for the detection of Android ransomware. They collected benign samples from AndroZoo dataset,[76] Android ransomware samples from the HelDroid dataset, and malware samples from Drebin, Contagio, and VirusTotal dataset. They analyzed dex files to extract static features of executable code such as system API packages, classes, and methods. They applied the RF algorithm for the multiclassification of Android ransomware, malware, and benign apps.

The authors in Reference 77 performed static analysis and employed an RF algorithm for the detection of Android ransomware. They collected benign samples from Play Store and Android ransomware samples from HelDroid, RansomProber, VirusTotal, and Koodous datasets. They analyzed dex files to extract API package calls. They compared the RF algorithm with DT, NB, and SVM algorithms where RF gave the best results for the detection of Android ransomware.

5.4 | Ensemble learning

Ensemble learning algorithm combines decisions from multiple models to build a strong model. It uses bagging (bootstrap aggregating) and boosting to combine models of the same category. In bagging, the models are parallel to each other to avoid the dependency and used stacking to combine

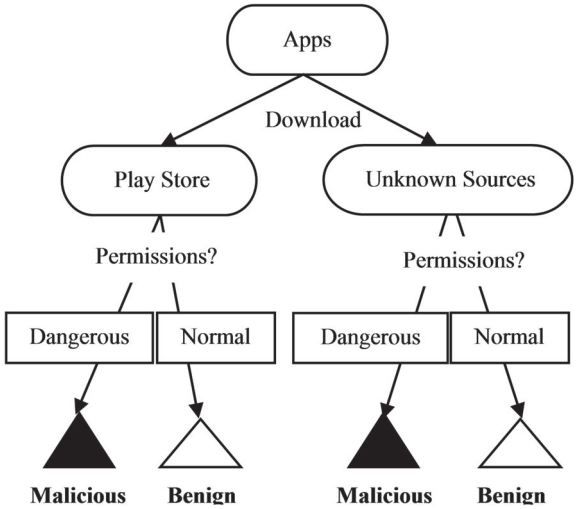

FIGURE 9 Decision tree example

different models (LR, DTs, RF, and SVM) by meta-classifier. Ensemble learning uses boosting to combine weighted results of weak algorithms. In boosting, the models are sequentially used with other models. For example, Adaptive Boosting (AdaBoost) algorithm.[78] Ensemble learning can learn the characteristics of Android apps based on their features and classify any unknown Android app into malicious or benign. Various models can be combined to create ensemble learning for the detection of Android ransomware which is discussed in the following:

5.4.1 | Logistic regression

Summary: LR is a binary classification algorithm in which the output labels are all either zero or one. For example, the algorithm will predict whether a given app is malicious or not. The output is calculated as follows:

$$\hat{y} = \frac{1}{1 + e^{-z}}(W^T X + b), \tag{4}$$

where,

- X is the features and b is bias.
- $\frac{1}{1+e^{-z}}$ is a sigmoid function such that the output should either be 0 or 1.
- W is an N-dimension vector and T is transpose.

5.4.2 | Support vector machine

Summary: SVM performs classification by finding an optimal solution that maximizes the margin between two classes. The margin is the distance between the hyperplane and closest data points (also known as support vectors) in each class.[79] SVM creates a line to separate 2D data, a plane to separate 3D data, and a hyperplane to separate higher-dimensional data. Figure 10 shows a linear SVM that finds a hyperplane to separate higher-dimensional data of malicious and benign apps with maximum margins by using support vectors. It automatically detects whether an unknown app falls on the malicious or benign side of the hyperplane.

Android ransomware detection: The authors in Reference 80 performed static analysis and employed ensemble learning for the detection of Android ransomware. They collected benign samples from Anzhi market[81] and ransomware samples from Chinese social networks. They performed analysis on permissions, system commands, window properties, and text present in AndroidManifest.xml files, decompiled code, layout files, and resource files. They combined LR, RF, DT, and SVM algorithms to create an ensemble learning method for the detection of Android locker ransomware.

The authors in Reference 82 performed static analysis and employed ensemble learning for the detection of Android ransomware. They collected benign samples from Play Store and ransomware samples from Contagio and VirusTotal datasets. They performed analysis on permissions, smali code, package names, URLs, text, and APK size. They applied gain ratio ranking to select important features out of all extracted features. They combined DT, RF, and stochastic gradient descent (SGD) algorithms to create an ensemble learning method for the detection of Android ransomware.

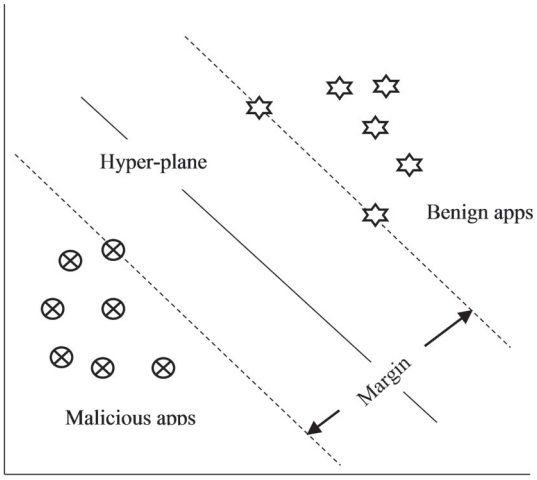

FIGURE 10 Support vector machine example

5.5 | Artificial neural networks

Summary: ANNs are motivated by the human brain, which consists of artificial neurons to perform the computations. Figure 11 shows a stacked neural network (also known as deep learning) that contains several hidden layers where features (Access_Picture, SMS_Read, and SIM_Card) represent three nodes in the input layer. These features are passed to the neurons present in the second layer, which further passed the generated output to the neurons present in the third layer. There are some randomly associated weights with features that are adjusted later to minimize the cost function by performing back-propagation.[83] The last layer provides the final value where output 0 implies a benign app, while output 1 implies a malicious app.

The layers present between the first (input) and last (output) layers are known as hidden layers. The hidden layers act as a black-box to perform the calculations. A node combines input from the data with a set of weights which are then summed and passed through an activation function. An artificial neuron uses association and activation functions in hidden layers. The association function is shown in the following equation:

$$z = \text{bias} + \Sigma_{i=1}^{N}(X_i \times W_i), \tag{5}$$

where,

- z is an association function.
- X_i are input features.
- W_i are weights.

An artificial neuron can use different activation functions in hidden layers such as Tan-h and ReLu. The activation functions are shown in the following equations:

$$\tanh = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{6}$$

$$\text{ReLu} = \max(0, z). \tag{7}$$

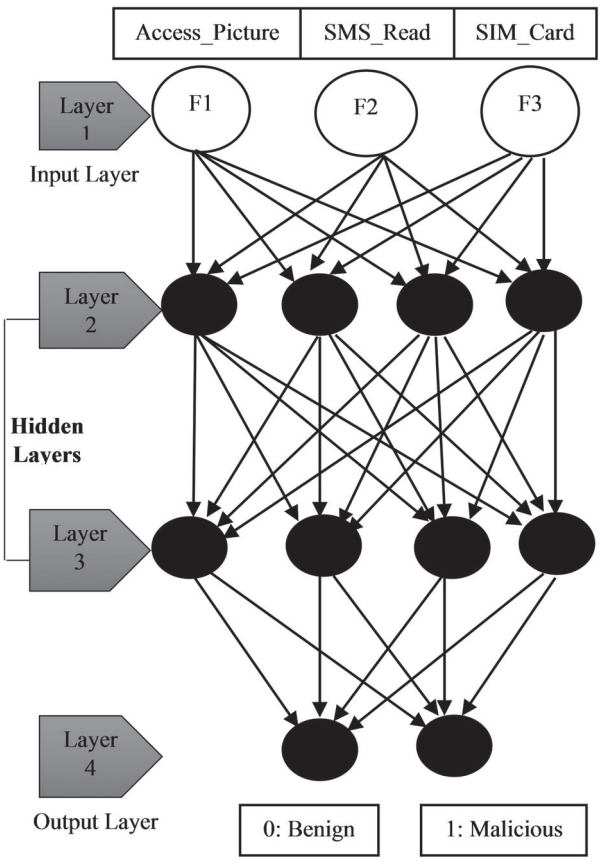

FIGURE 11 Artificial neural networks example

But the exception lies in the output layer where Sigmoid activation function is used for binary classification where output is either 0 or 1. The Sigmoid activation function is shown in the following equation.

$$\text{Sigmoid} = \frac{1}{1+e^{-z}}. \tag{8}$$

Android ransomware detection: The authors in Reference 84 performed hybrid analysis and employed deep neural network (DNN) for the detection of Android ransomware. They collected benign samples from Play Store and Android ransomware samples from R-PackDroid, HelDroid, and Contagio datasets. They performed static analysis on permissions, API methods, text (monetary, threat, porn, lock, encrypt), and images (logos, nudity) after disassembling the APK files. They also performed dynamic analysis on system and API calls sequence by running each sample in a sandbox to identify the infected API calls. They applied NB, SVM, RF, AdaBoost, and DNN to detect Android ransomware. The results showed that DNN gave the best results in comparison to the other algorithms for the detection of Android ransomware.

The authors in Reference 85 performed static analysis and employed ANN for the detection of Android ransomware. They collected benign samples from the AndroZoo dataset and Android ransomware samples from the RansomProber dataset. They performed static analysis on permissions, intents, strings extracted from text and images, locking, encryption, and encoding methods after disassembling the APK files. They applied LR, SVM, and ANN to detect Android ransomware. The results showed that ANN gave the best results in comparison to the other algorithms for the detection of Android ransomware.

Discussion: This section explained machine learning techniques for the detection of Android ransomware. Table 6 presents the analysis methods and detection techniques implemented by researchers to detect the ransomware attack on Android OS. The following section gives answers to formulated research questions.

6 | DISCUSSION

This section provides answers to the research questions formulated in Section 1 (refer to Table 1).

6.1 | RQ1: Which analysis method is widely adopted by the researchers for feature extraction of Android ransomware?

The static analysis method extracts features by analyzing the program code of ransomware without running it in a virtual environment. But the attackers can evade this method by performing code obfuscation and this method fails to obtain the run-time features of Android ransomware. In that case, the dynamic analysis method extracts run-time features by executing the program code of ransomware in an emulator. But some of the executing paths may get missed in dynamic analysis. In addition, the attackers can detect the virtual environment after analyzing the nonavailability of GPS and gyroscope[91] sensors. Moreover, the required period to analyze malicious behavior is not defined in the dynamic analysis method.[92] Therefore, the hybrid analysis method implements both static and dynamic analysis methods to extract static as well as dynamic features of Android ransomware. The importance of using hybrid analysis is that it increases code coverage and finds more features as compared with using static and dynamic analysis methods individually.[5]

Table 6 shows the analysis methods adopted by various researchers for feature extraction of Android ransomware. From this table, we analyze that the researchers in literature widely adopted static analysis followed by hybrid, and dynamic analysis methods. Notably, none of the researchers in the literature adopted the pseudo-dynamic analysis method (refer to Section 6.4.4) to eliminate the drawbacks of static and dynamic analysis.

6.2 | RQ2: Which features have been extracted by the researchers after analyzing Android ransomware?

It can be observed from Table 6 that the researchers extracted features after applying static/ dynamic/ hybrid analysis methods on Android ransomware samples. The researchers analyzed AndroidManifest.xml, JAR, dex, layout, resource, and smali files to extract features using static analysis. The static features extracted by the researchers in the existing literature include permissions, intents, JAVA bytecode, Dalvik bytecode, smali code, locking, encryption, text, opcode sequence, package names, URLs, APK size, window properties, system commands, classes, API packages, native instructions, and encoding methods. The dynamic features extracted by the researchers include a processor, memory, input-output usages, widget, activity, texts, buttons, and system calls. Besides this, the hybrid features extracted by the researchers include locking, encryption, threatening text, network traffic, permissions, API call sequence, system paths, malicious domain names, user interactions, opcodes, memory, CPU usage, images of logos, call graph, and encryption methods.

TABLE 6 Summary of Android ransomware analysis and detection methods

Reference	Framework	Year	Analysis methods	Analyzed files	Extracted features	Detection techniques	Classifications algorithms
51	HelDroid	2015	Hybrid	Smali, Resource	Locking, Encryption, Threatening text, Network traffic	Natural language processing	X
64	ADAAR	2015	Hybrid	AndroidManifest.xml, Smali	Permissions, API call sequence, System paths, Malicious domain names	X	X
54	RIO	2016	Static	AndroidManifest.xml	Permissions, Intents	Logic rules	X
58	RSP	2016	Static	JAR	JAVA bytecode	Logic rules	X
69	HMM-based	2016	Static	Smali	Opcodes sequence	Machine learning	HMM
61	RPT	2016	Dynamic	X	Processor, Memory, Input-output usages	Process monitoring	X
71	MADAM	2016	Hybrid	AndroidManifest.xml	System calls, Permissions, User interactions	Machine learning	*k*NN
86	Opcode-based	2017	Static	Dex	Opcode sequence	Machine learning	LDA
57	R-PackDroid	2017	Static	Dex	Dalvik bytecode	Machine learning	RF
82	GreatEatlon	2017	Static	AndroidManifest.xml, Resource	Permissions, Smali code, Text, Package names, URLs, APK size	Machine learning	DT, RF, SGD, ensemble learning
74	Extinguishing Ransomware	2017	Hybrid	Smali	Opcodes, CPU usage, Network traffic logs, System calls	Machine learning	DT, LR, NB
84	DNA-Droid	2017	Hybrid	AndroidManifest.xml, Resource	Permissions, Text, Images of logos, System and API calls	Machine learning	NB, SVM, RF, DNN
59	SEC	2018	X	Dex	X	Structural Entropy	X
80	Locker-Ransomwar	2018	Static	AndroidManifest.xml, Resource	Permissions, Text, Intents, Window properties, System commands	Machine learning	SVM, DT, RF, LR, ensemble learning
87	Encryption-based	2018	Static	Dex	Encrption functions	X	X
88	Talos	2018	Static	Dex	Classes, Methods	X	Calculus of communicating systems
52	RansomProber	2018	Dynamic	X	Widget, Activity, Texts, Buttons	User interaction	X
65	RanDroid	2018	Hybrid	Layout, Dex, Resource	Threatening text, Images	User interaction	X
75	API-based	2019	Static	Dex	API packages, Classes, Methods	Machine learning	RF
72	Native-based	2019	Static	Native code	Native instructions	Machine learning	*k*NN, RF, ANN, SVM, NB
77	API-RDS	2019	Static	Dex	API calls	Machine learning	RF, DT, NB, SVM
62	SystemCall-based	2019	Hybrid	AndroidManifest.xml, Dex	Permissions, Opcodes, APIs, System Calls, Call graph, Network trace	Machine learning	RF, AdaBoost
66	RTARD	2019	Hybrid	Layout	APIs, Encryption, System calls	Shannon entropy	X
55	EoP	2020	Static	AndroidManifest.xml	Permissions	Machine learning	RF, DT, NB
56	RansomAnalysis	2020	Static	AndroidManifest.xml	Permissions, Intents	X	X
89	Extreme Learning	2020	Static	AndroidManifest.xml, Dex	Permissions, API calls	Machine Learning	Kernel Extreme Learning Machine
85	GPU-based	2020	Static	AndroidManifest.xml, Resource, JAR	Permissions, Intents, Text, Images, Locking, Encryption, Encoding	Machine Learning	ANN, LR, SVM
90	Dynamic-based	2020	Dynamic	X	System calls	Machine Learning	RF, DT, NB

Abbreviations: ADAAR, automated detection and analysis for Android ransomware; ANN, artificial neural network; API, application programming interface; APK, Android package kit; DNN, deep neural network; DT, decision tree; EoP, effectiveness of permissions; HMM, hidden Markov model; LR, logistic regression; NB, Naive Bayes; RF, random forest; RIO, ransomware inside out; RPT, ransomware prevention technique; RSP, ransomware steals your phone; SGD, stochastic gradient descent; SVM, support vector machine.

These features are important to analyze as it further acts as input to various machine learning algorithms for the detection of Android ransomware. The detection accuracy of machine learning algorithms heavily depends on the extracted features. Thus, significant features can be further extracted (refer to Section 6.4.3) to accurately detect Android ransomware.

6.3 | RQ3: Which detection technique is widely adopted in the existing literature to detect Android ransomware?

It can be examined from Table 6 that most of the researchers employed machine learning techniques for the detection of Android ransomware. The researchers in the existing literature widely employed RF followed by DT, NB, SVM, LR, ANN, and ensemble learning algorithms. On the other hand, machine learning algorithms such as *k*NN, HMM, LDA, and SGD are less frequently employed by the researchers in the existing literature. Notably, out of all machine learning techniques, the researchers achieved the best detection accuracy by employing ensemble learning algorithms including the RF algorithm. The ensemble learning algorithm is the strongest approach because it combines decisions from multiple algorithms to build an algorithm that achieves better performance than an individual machine learning algorithm.[80,82]

None of the researchers in the existing literature employed unsupervised machine learning algorithms for detection of Android ransomware (refer Section 6.4.2). These algorithms require no historical labels and can efficiently detect unknown Android ransomware samples too. Moreover, the deep learning algorithms (refer to Section 6.4.2) can be further employed to detect Android ransomware.

6.4 | RQ4: What are the research challenges in the field of Android ransomware?

A thorough study of analysis and detection of Android ransomware reveals the following research challenges with observations and suggestions to fill the gaps and carry out further research:

6.4.1 | Datasets

Observations: In literature, RansomProber[52] and HelDroid[51] datasets provide 2300 and 670 Android ransomware APKs, respectively. But the researchers require updated and large datasets specifically for Android ransomware to perform the experimental work. There is a law of machine learning—"The more data an algorithm can train on, the more accurate it will be." Deep learning, which is a part of machine learning obeys this law and provides more detection accuracy (given a large amount of data) than traditional machine learning algorithms. For example, the authors in Reference 93,94 implemented deep learning algorithms for the detection of generic Android malware and achieved better results as compared with traditional machine learning algorithms. Thus, the deep learning algorithms (with large datasets) can be used for the detection of Android ransomware to achieve better results.

Suggestions: For a large dataset, there is a requirement of deployment of honey-pots on smart-phones to collect samples of Android ransomware. The collected datasets can further act as an input for various deep learning algorithms which will be discussed in the next point 6.4.2.

6.4.2 | Detection methods

Observations: In literature, the researchers used traditional supervised machine learning algorithms for the detection of Android ransomware, as shown in Table 6. The authors in Reference 84 employed DNN with three hidden layers for the detection of Android ransomware where the dataset contains 1900 malicious and 2500 benign samples approximately. However, as discussed in previous point 6.4.1, the available datasets of Android ransomware are not enough to employ deep learning algorithms with a large number of hidden layers. Moreover, the supervised machine learning algorithms are already familiar with the targets (i.e., whether the given sample is Android ransomware or not); and are not able to detect unknown Android ransomware samples in the wild.

Suggestions: After collecting Android ransomware datasets, the researchers can employ deep learning algorithms such as recurrent neural network (RNN), long short-term memory units (LSTM), convolutional neural network (CNN), and so forth. RNN not only takes present input but also previously generated output (recent past) to perform computations. But it faces a vanishing gradient problem because the information flowing in RNN passes through several levels of multiplication. This problem is addressed by LSTM, which allows RNN to continue its learning over several steps. LSTM exchanges the units in the hidden layer of RNN with the gated cells from where the information can be stored and read.[95] But Android ransomware can display threatening text via fake images on users' smart-phone screen to evade from being detected. In this case, CNN can be employed to perform text analysis on fake images via optical character recognition where input images are symbols to be transcribed.[96]

In addition, unsupervised machine learning algorithms including *k*-means and Gaussian Mixture Models can be employed for detection of Android ransomware. The unsupervised machine learning algorithms require no historical labeling and care able to detect unknown Android ransomware samples too.

6.4.3 | Features

Observations: In literature, the researchers performed static, dynamic, and hybrid analysis methods for feature extraction of Android ransomware. The extracted features as shown in Table 6 include permissions, intents, encryption, locking, threatening text, and so forth. However, there is a requirement to extract more features of Android ransomware, which can act as an input for deep learning algorithms.

Suggestions: The Android ransomware samples communicate with the attacker to transfer users' personal information from the smart-phone to the C&C server. Thus, the network traffic can be analyzed to extract features such as the size of a packet, the number of packets sent/ received per unit time, suspicious IP addresses, ports, and so forth. More analysis is needed towards library files to extract native machine code and call graphs.

6.4.4 | Analysis methods

Observations: In literature, the researchers used static and dynamic analysis methods to extract features of Android ransomware. However, these methods suffer from several drawbacks (refer to RQ1 6.1). The hybrid analysis method does not combine static and dynamic methods to create a new method but applies static analysis followed by dynamic analysis method to extract a large number of features of Android ransomware. Thus, the hybrid analysis combines the advantages of static and dynamic analysis methods; but does not eliminate their drawbacks.

Suggestions: The pseudo-dynamic analysis method can overcome the drawbacks of static and dynamic analysis methods. The pseudo-dynamic analysis does not execute the code in a virtual environment but follows the instructions in a way such that it replicates the execution of the Android ransomware. For example, the authors in Reference 97 employed pseudo-dynamic analysis where they logged and tracked API calls to analyze generic Android malware without executing it.

7 | CONCLUSIONS

Earlier the ransomware attack was limited to PC but due to the rapid growth of Android smart-phones and its increasing usage in the real-world; cyber-criminals are now targeting it with ransomware to get personal information for illegal monetary gains. Throughout the literature, the researchers performed a survey to analyze generic Android malware. Hence, we observed that there was a need to survey the analysis and detection methods of Android ransomware.

In this article, we attempted to present a threat scenario of the ransomware attack on Android OS. A comprehensive review was done by giving example papers on analysis and detection methods of Android ransomware from its beginning (2015) till date (2020). The researchers widely used static analysis methods for feature extraction and supervised machine learning techniques for the detection of Android ransomware. However, the biggest gap noticed is the requirement of Android ransomware datasets. The deep learning algorithms can be employed using a large dataset to achieve better results for the detection of Android ransomware. The unsupervised machine learning techniques do not require historical labeling and can be employed to detect unknown Android ransomware samples. Finally, we gave answers to the research questions and pointed out the observations and suggestions for researchers and practitioners, to carry out further research towards detection of Android ransomware.

DATA AVAILABILITY STATEMENT

The HelDroid[51] and RansomProber[52] datasets that support the findings of this study are available from the authors. Restrictions apply to the availability of these data, which were used under license for this study. Data are available from the authors with permission via sending an application email.

ORCID

Shweta Sharma https://orcid.org/0000-0002-7842-3471

REFERENCES

1. La Polla M, Martinelli F, Sgandurra D. A survey on security for mobile devices. *IEEE Commun Surv Tutor*. 2013;15(1):446-471.
2. Faruki P, Bharmal A, Laxmi V, et al. Android security: a survey of issues, malware penetration, and defenses. *IEEE Commun Surv Tutor*. 2014;17(2):998-1022.
3. Tan DJ, Chua TW, Thing VL. Securing Android: a survey, taxonomy, and challenges. *ACM Comput Surv*. 2015;47(4):1-45.

4. Xu M, Song C, Ji Y, et al. Toward engineering a secure Android ecosystem: a survey of existing techniques. *ACM Comput Surv*. 2016;49(2):1-47.
5. Tam K, Feizollah A, Anuar NB, Salleh R, Cavallaro L. The evolution of Android malware and Android analysis techniques. *ACM Comput Surv*. 2017;49(4):1-41.
6. Komatwar R, Kokare M. A survey on malware detection and classification. *J Appl Secur Res*. 2020;1-31.
7. Qiu J, Zhang J, Luo W, Pan L, Nepal S, Xiang Y. A survey of Android malware detection with deep neural models. *ACM Comput Surv*. 2020;53(6):1-36.
8. Lipovsky R, Stefanko L, Branisa G. The rise of Android ransomware; 2016. https://www.welivesecurity.com/wp-content/uploads/2016/02/Rise_of_Android_Ransomware.pdf. Accessed May 31, 2018.
9. Statista Number of smartphone users worldwide from 2016 to 2021 (in billions); 2020. . https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide//; Accessed December 7, 2020.
10. Felt AP, Finifter M, Chin E, Hanna S, Wagner D. A survey of mobile malware in the wild. Paper presented at: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices; 2011:3-14; Chicago, IL.
11. McAffee Labs threats report; 2020. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf. [Online; Accessed December 8, 2020].
12. Symantec Internet security threat report; 2019. https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf. [Online; Accessed December 9, 2020].
13. Netmarketshare Mobile operating system market share; 2020. https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1&qpcd=1. [Online; Accessed December 9, 2020].
14. Developers A Codenames, tags, and build numbers; 2020. https://source.Android.com/setup/start/build-numbers. [Online; Accessed February 03, 2020].
15. cnet Best Android phones for 2019; 2019. https://www.cnet.com/topics/phones/best-phones/android/. [Online; Accessed July 16, 2019].
16. Apple iPhone; 2019. https://www.apple.com/in/iphone/. [Online; Accessed July 16, 2019].
17. cnet Best windows phones; 2018. https://www.cnet.com/topics/phones/best-phones/windows/. [Online; Accessed July 16, 2019].
18. Segan S. The 10 best symbian phones ever; 2013. http://www.pcmag.com/feature/307321/the-10-best-symbian-phones-ever. [Online; Accessed July 16, 2019].
19. BlackBerry The blackberry phones; 2019. https://global.blackberry.com/en/smartphones. [Online; Accessed July 16, 2019].
20. Developers A Android architecture; 2020. https://source.android.com/devices/architecture. [Online; Accessed February 02, 2020].
21. Developers A Platform architecture; 2020. https://developer.android.com/guide/platform. [Online; Accessed February 02, 2020].
22. Elenkov N. Android's security model; 2014. https://www.nostarch.com/download/Android_Security_Internals_ch1.pdf. Online; Accessed November 25, 2018.
23. Android Permission; 2019. https://developer.android.com/guide/topics/manifest/permission-element.html. [Online; Accessed May 25, 2019].
24. Developers A Android 5.0 behavior changes; 2020. https://developer.android.com/about/versions/android-5.0-changes#custom_permissions. [Online; Accessed February 03, 2020].
25. Developers A The fundamental Android security models; 2020. https://medium.com/modulotech/the-fundamental-android-security-models-64f07dda006a. [Online; Accessed February 05, 2020].
26. Developers A. System and kernel security; 2020. https://source.android.com/security/overview/kernel-security. [Online; Accessed February 05, 2020].
27. Android Normal permissions; 2019. https://developer.android.com/guide/topics/permissions/normal-permissions.html. [Online; Accessed May 21, 2019].
28. Developers Requesting permissions; 2019. https://developer.android.com/guide/topics/permissions/requesting.html. [Online; Accessed May 21, 2019].
29. Wong A. Marshmallow permission level; 2017. https://gist.github.com/andhie/70098e17ded91eee19e9. [Online; Accessed May 21, 2019].
30. Rawnsley R. Android permissions - protection levels; 2011. http://rupertrawnsley.blogspot.in/2011/11/android-permissions-protection-levels.html. [Online; Accessed May 21, 2019].
31. Tetali SD. Keeping 2 billion Android devices safe with machine learning; 2018. https://android-developers.googleblog.com/2018/05/keeping-2-billion-android-devices-safe.html. [Online; Accessed February 05, 2020].
32. GooglePlay <Fast Charging>; 2018. https://play.google.com/store/apps/details?id=chargerbooster.charger.faster.booster. [Online; Accessed May 09, 2018].
33. GooglePlayStore <DU Battery Saver - Battery Charger & Battery Life>; 2018. https://play.google.com/store/apps/details?id=com.dianxinos.dxbs. [Online; Accessed May 09, 2018].
34. PlayStore <SMS Blocker - Clean Inbox>; 2018. https://play.google.com/store/apps/details?id=com.smsBlocker. [Online; Accessed May 09, 2018].
35. Fang Z, Han W, Li Y. Permission based Android security: issues and countermeasures. *Comput Secur*. 2014;43:205-218.
36. Shin W, Kwak S, Kiyomoto S, Fukushima K, Tanaka T. A small but non-negligible flaw in the Android permission scheme. Paper presented at: Proceedings of the IEEE International Symposium on Policies for Distributed Systems and Networks; 2010:107-110; Fairfax, VA.
37. Fedler R, Kulicke M, Schütte J. Native code execution control for attack mitigation on Android. Paper presented at: Proceedings of the 3rd ACM Workshop on Security and Privacy in Smartphones & Mobile Devices; 2013:15-20; Berlin, Germany.
38. Popoola SI, Ojewande SO, Sweetwilliams FO, John S, Atayero A. Ransomware: current trend, challenges, and research directions. Paper presented at: Proceedings of the Interanational Multiconference of Engineers and Computer Scientists, Hong Kong: IEEE; 2017:169-174.
39. Yalew SD, Maguire GQ, Haridi S, Correia M. Hail to the thief: protecting data from mobile ransomware with ransomsafedroid. Paper presented at: Proceedings of the International Symposium on Network Computing and Applications (NCA), Cambridge: IEEE; 2017:1-8.
40. Unuchek R. Dvmap: the first Android malware with code injection; 2017. https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/. [Online; Accessed January 2, 2018].
41. Point C. KSN Report Ransomware in 2014 - 2016; 2016. https://securelist.com/analysis/publications/75183/ksn-report-mobile-ransomware-in-2014-2016/. [Online; Accessed May 31, 2018].
42. Buchka N, Kivva A. Disassembling a mobile Trojan attack; 2016. https://securelist.com/disassembling-a-mobile-trojan-attack/76286/. [Online; Accessed July 11, 2018].
43. Symantec Android.Lockdroid.E; 2014. https://www.symantec.com/security_response/writeup.jsp?docid=2014-103005-2209-99. [Online; Accessed July 11, 2018].

44. Zheng C, Xiao C, Xu Z. New Android Trojan 'Xbot' phishes credit cards and bank accounts, encrypts devices for ransom; 2016. https://researchcenter.paloaltonetworks.com/2016/02/new-android-trojan-xbot-phishes-credit-cards-and-bank-accounts-encrypts-devices-for-ransom/ [Online; Accessed 11-Jul-2018].
45. Lipovsky R, Stefanko L, Branisa G. Trends in android malware; 2016. https://www.welivesecurity.com/wp-content/uploads/2017/02/ESET_Trends_2017_in_Android_Ransomware.pdf. [Online; Accessed May 31, 2018].
46. Yuan L. Android.WannaLocker; 2017. https://www.symantec.com/security_response/writeup.jsp?docid=2017-061314-4839-99&tabid=2. [Online; Accessed July 11, 2018].
47. ESET Android.WannaLocker; 2017. https://www.welivesecurity.com/2017/10/13/doublelocker-innovative-android-malware/. [Online; Accessed July 2, 2019].
48. Ducklin P. Android malware uses coronavirus for sextortion and ransomware combo; 2020. https://nakedsecurity.sophos.com/2020/03/18/android-malware-uses-coronavirus-for-sextortion-ransomware-combo/. [].
49. Mix New Android ransomware threatens to 'expose' your porn escapades to the FBI; 2020. https://thenextweb.com/security/2020/04/28/android-ransomware-fbi-porn/. [].
50. Enck W, Gilbert P, Han S, et al. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans Comput Syst*. 2014;32(2):1-15.
51. Andronio N, Zanero S, Maggi F. Heldroid: dissecting and detecting mobile ransomware. Paper presented at: Proceedings of the 18th International Symposium on Recent Advances in Intrusion Detection; 2015:382-404; Kyoto, Japan.
52. Chen J, Wang C, Zhao Z, Chen K, Du R, Ahn GJ. Uncovering the face of android ransomware: characterization and real-time detection. *IEEE Trans Inf Forens Secur*. 2018;13(5):1286-1300.
53. Sharma S, Kumar N, Kumar R, Krishna CR. The paradox of choice: investigating selection strategies for android malware datasets using a machine-learning approach. *Commun Assoc Inf Syst*. 2020;46(1):26.
54. Mercaldo F, Nardone V, Santone A. Ransomware inside out. Paper presented at: Proceedings of the 11th IEEE International Conference on Availability, Reliability and Security, Salzburg, Austria; 2016:628-637.
55. Alsoghyer S, Almomani I. On the effectiveness of application permissions for Android ransomware detection. Paper presented at: Proceedings of the2020 6th Conference on Data Science and Machine Learning Applications (CDMA), Riyadh, Saudi Arabia: IEEE; 2020:94-99.
56. Sharma S, Kumar R, Krishna CR. RansomAnalysis: the evolution and investigation of android ransomware. Paper presented at: Proceedings of the International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR; 2020:33-41; Chandigarh, India; Springer.
57. Maiorca D, Mercaldo F, Giacinto G, Visaggio CA, Martinelli F. R-PackDroid: API package-based characterization and detection of mobile ransomware. Paper presented at: Proceedings of the Symposium on Applied Computing; 2017:1718-1723; Marrakech, Morocco.
58. Mercaldo F, Nardone V, Santone A, Visaggio CA. Ransomware steals your phone. formal methods rescue it. Paper presented at: Proceedings of the 11th IEEE International Conference on Availability, Reliability and Security; 2016:212-221; Heraklion, Greece.
59. Cuzzocrea A, Martinelli F, Mercaldo F. A novel structural-entropy-based classification technique for supporting android ransomware detection and analysis. Paper presented at: Proceedings of the IEEE 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE); 2018:1-7.
60. Egele M, Scholte T, Kirda E, Kruegel C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput Surv*. 2012;44(2):1-49.
61. Song S, Kim B, Lee S. The effective ransomware prevention technique using process monitoring on android platform. *Mob Inf Syst*. 2016;2016:1–9.
62. Deepa K, Radhamani G, Vinod P, Shojafar M, Kumar N, Conti M. Identification of Android malware using refined system calls. *Concurr Comput Pract Exper*. 2019;30(11):1-24.
63. Lemmou Y, Lanet JL, Souidi EM. A behavioural in-depth analysis of ransomware infection. *IET Inf Secur*. 2020;15(1):38-58.
64. Yang T, Yang Y, Qian K, Lo DCT, Qian Y, Tao L. Automated detection and analysis for Android ransomware. Paper presented at: Proceedings of the 17th IEEE International Conference on High Performance Computing and Communications; 2015:1338-1343; New York.
65. Alzahrani A, Alshehri A, Alshahrani H, et al. RanDroid: structural similarity approach for detecting ransomware applications in Android platform. Paper presented at: Proceedings of the 18th IEEE International Conference on Electro/Information Technology, Rochester, Michigan; 2018:0892-0897.
66. Ko JS, Jo JS, Kim DH, Choi SK, Kwak J. Real time Android ransomware detection by analyzed Android applications. Paper presented at: Proceedings of the 18th IEEE International Conference on Electronics, Information, and Communication, Auckland, New Zealand; 2019:1-5.
67. Buczak AL, Guven E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Commun Surv Tutor*. 2016;18(2):1153-1176.
68. Maloof MA. *Machine Learning and Data Mining for Computer Security: Methods and Applications*. New York, NY: Springer; 2006.
69. Canfora G, Mercaldo F, Visaggio CA. An hmm and structural entropy based detector for Android malware: an empirical study. *Comput Secur*. 2016;61:1-18.
70. Zhao X, Zhang S. A review on facial expression recognition: feature extraction and classification. *IETE Tech Rev*. 2016;33(5):505-517.
71. Saracino A, Sgandurra D, Dini G, Martinelli F. Madam: effective and efficient behavior-based Android malware detection and prevention. *IEEE Trans Depend Sec Comput*. 2016;15(1):83-97.
72. Lachtar N, Ibdah D, Bacha A. The case for native instructions in the detection of mobile ransomware. *IEEE Lett Comput Soc*. 2019;2(2):16-19.
73. Guan D, Yuan W. A survey of mislabeled training data detection techniques for pattern classification. *IETE Tech Rev*. 2013;30(6):524-530.
74. Ferrante A, Malek M, Martinelli F, Mercaldo F, Milosevic J. Extinguishing ransomware-a hybrid approach to Android ransomware detection. Paper presented at: Proceedings of the 10th International Symposium on Foundations and Practice of Security; 2017:242-258; Springer, New York, NY.
75. Scalas M, Maiorca D, Mercaldo F, Visaggio CA, Martinelli F, Giacinto G. On the effectiveness of system API-related information for Android ransomware detection. *Comput Secur*. 2019;86:168–182.
76. Li L, Gao J, Hurier M, et al. <AndroZoo++: collecting millions of Android apps and their metadata for the research community>; 2017. https://arxiv.org/abs/1709.05281. [Online; Accessed April 25, 2018].
77. Alsoghyer S, Almomani I. Ransomware detection system for Android applications. *Electronics*. 2019;8(8):868.
78. Hu W, Hu W, Maybank S. Adaboost-based algorithm for network intrusion detection. *IEEE Trans Syst Man Cybern B Cybern*. 2008;38(2):577-583.
79. Ye Y, Li T, Adjeroh D, Iyengar SS. A survey on malware detection using data mining techniques. *ACM Comput Surv*. 2017;50(3):1-40.
80. Su D, Liu J, Wang X, Wang W. Detecting Android locker-ransomware on Chinese social networks. *IEEE Access*. 2018;7:20381-20393.
81. AnzhiMarket; 2019. http://www.anzhi.com/. [Online; Accessed August 2, 2019].

82. Zheng C, Dellarocca N, Andronio N, Zanero S, Maggi F. Greateatlon: Fast, static detection of mobile ransomware. Paper presented at: Proceedings of the International Conference on Security and Privacy in Communication Systems; 2017:617-636; Springer, New York, NY.
83. Venketesh P, Venkatesan R. A survey on applications of neural networks and evolutionary techniques in web caching. *IETE Tech Rev*. 2009;26(3):171-180.
84. Gharib A, Ghorbani A. DNA-Droid: a real-time Android ransomware detection framework. Paper presented at: Proceedings of the International Conference on Network and System Security; 2017:184-198; Springer, New York, NY.
85. Sharma S, Krishna CR, Kumar R. Android ransomware detection using machine learning techniques: a comparative analysis on GPU and CPU. Paper presented at: Proceedings of the International Arab Conference on Information Technology (ACIT), Misr University for Science & Technology, Giza, Egypt: IEEE; 2020:1-6.
86. Karimi A, Moattar MH. Android ransomware detection using reduced opcode sequence and image similarity. Paper presented at: Proceedings of the International Conference on Computer and Knowledge Engineering (ICCKE), Ferdowsi University of Mashhad, Iran: IEEE; 2017:229-234.
87. Mercaldo F, Di Sorbo A, Visaggio CA, Cimitile A, Martinelli F. An exploratory study on the evolution of Android malware quality. *J Softw Evolut Process*. 2018;30(11):1-22.
88. Cimitile A, Mercaldo F, Nardone V, Santone A, Visaggio CA. Talos: no more ransomware victims with formal methods. *Int J Inf Secur*. 2018;17(6):719-738.
89. Faris H, Habib M, Almomani I, Eshtay M, Aljarah I. Optimizing extreme learning machines using chains of salps for efficient Android ransomware detection. *Appl Sci*. 2020;10(11):3706.
90. Abdullah Z, Muhadi FW, Saudi MM, Hamid IRA, Foozy CFM. Android ransomware detection based on dynamic obtained features. Paper presented at: Proceedings of theInternational Conference on Soft Computing and Data Mining; 2020:121-129; Springer, New York, NY.
91. Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, Ioannidis S. Rage against the virtual machine: hindering dynamic analysis of Android malware. Paper presented at: Proceedings of the 7th ACM European Workshop on System Security; 2014:1-6; Amsterdam, Netherlands.
92. Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y. "Andromaly": a behavioral malware detection framework for Android devices. *J Intell Inf Syst*. 2012;38(1):161-190.
93. Yuan Z, Lu Y, Xue Y. Droiddetector: Android malware characterization and detection using deep learning. *Tsinghua Sci Technol*. 2016;21(1):114-123.
94. Li D, Zhao L, Cheng Q, Lu N, Shi W. Opcode sequence analysis of Android malware by a convolutional neural network. *Concurr Comput Pract Exper*. 2019;32(18):e5308
95. DL4J A Beginner's guide to recurrent networks and LSTMs; 2017. https://deeplearning4j.org/lstm. [Online; Accessed November 05, 2017].
96. AlexDBlack Convolutional networks; 2019. https://deeplearning4j.org/cn/convolutionalnets. [Online; Accessed July 12, 2019].
97. Nix R, Zhang J. Classification of Android apps and malware using deep neural networks. Paper presented at: Proceedings of the IEEE International Joint Conference on Neural Networks; 2017:1871-1878; Anchorage, AK.

How to cite this article: Sharma S, Kumar R, Rama Krishna C. A survey on analysis and detection of Android ransomware. *Concurrency Computat Pract Exper*. 2021;33:e6272. https://doi.org/10.1002/cpe.6272