Received: 31 May 2019 | Revised: 27 August 2019 | Accepted: 5 September 2019

DOI: 10.1002/ett.3773

SPECIAL ISSUE ARTICLE

WILEY

Privacy issues of android application permissions: A literature review

Gulshan Shrivastava[1] | **Prabhat Kumar**[1] | **Deepak Gupta**[2] | **Joel J. P. C. Rodrigues**[3,4]

[1]Department of Computer Science and Engineering, National Institute of Technology, Patna, India

[2]Computer Science and Engineering, Maharaja Agrasen Institute of Technology, New Delhi, India

[3]Department of Electrical Engineering, Federal University of Piauí (UFPI), Teresina, PI, Brazil

[4]Covilhã Delegation, Instituto de Telecomunicações, Covilhã, Portugal

Correspondence
Gulshan Shrivastava, Department of Computer Science and Engineering, National Institute of Technology Patna, Ashok Rajpath, Patna-800 005, Bihar, India.
Email: gulshanstv@gmail.com

Funding information
National Institute of Technology Patna, India; Fundação para a Ciência e a Tecnologia (FCT), Grant/Award Number: UID/EEA/50008/2019; Brazilian National Council for Research and Development, Grant/Award Number: 309335/2017-5

Abstract

Android is an application platform for mobile devices. It comprises of the operating system, software framework, and core programs. This platform uses permissions to hide precious information about the user from untrusted apps. However, to install an application, device feature uses permissions that are granted by the user. User has the ability to analyze permissions and abort the setup if the permissions are unfriendly or unrestrained. Android permission analysis schemes show a significant role to fight against these undesirable behaviors of untrusted android apps in the aspect of security and privacy. This survey attempts to deal with the android application permissions that are related security and privacy challenges. It includes various research articles published in computers and security, digital investigation, decision support systems, systems and software security, and information forensics journals in the last 10 years. The survey is based on the following considerations: research issues motivated by the scheme, the methodology used, ability of result analysis conducted, and android features considered for performance evaluation.

1 | INTRODUCTION

The increasing use of the smartphones in the daily life has encouraged the mobile industry for the regeneration.[1] As reported in the works of Powar and Meshram[2] and Felt et al,[3] the Android mobile framework underwent a transformation by the manufacturing industry, customers, and the software extension community. In 2015, the official Android market (Google Play) reported over 1.5 million users downloaded the app,[4] thus accounting for over a billion Android customers. Android is the most used mobile device operating system (OS) at present.[5] One of the primary reason for its rise is its availability as freeware OS. The earlier Android gadgets did not accept security patches, which leads to less number of users, but successive improvement[6-8] has solved the problem.

Android is the most popular OS at present. Android has customized our personal life as it finds in applications in a wide range of gadgets, like mobile phones and televisions. In the play store, there are over 2.5 billion active android users.[9] The Google has provided open source licenses, and the available source code is the reason behind the popularity of Android. Android is well known with the mobile organizations that require an instant, adaptable OS for their devices.[10,11]

Mobile device security is utmost important concern, but it also should be user friendly. However, it is the very fine ensuring a client's information protection along with the security. The mobile device companies are concerned regarding a user's security and data privacy. The internet creates complex issues for securing user's data privacy that handles threats and vulnerabilities.[12-14] The data is instantly transferred from infected phones to premium-rate phone numbers without even knowledge of the client. Because of this, the individuals' data is being sent to the unauthorized third parties.[15-17] This paper supplements the previous reviews by extending the scope of malware development and Android security issues. The permissions given by the defined smartphone are checked and compared, and these permissions are granted accordingly.

This work consists of detailed literature review. Sections 2 and 3 discuss the background literature of permission analysis and its implementation in android. In Section 4, attacks are presented through permission techniques, and consequently, Section 5 deals with the methodology of permission analysis. Section 6 analyzes the approach for the literature review. In Sections 7 and 8, exclusion criteria and data extraction are studied, respectively. Section 9 discusses future research directions, and the last section finally conclude this paper.

2 | BACKGROUND INFORMATION ON PERMISSION ANALYSIS

Malware detection on smartphones can be separated into static and dynamic types. In static detection, some features are removed from an app and analyzed before the app is executed,[18-22] while in case dynamic detection, the app is executed via simulator and decided according to the log files.[23-27] Both types have some advantages and disadvantages as stated in Table 1.

2.1 | Static analysis

Static analysis is a way of detecting malicious behavior in the code segments. Static analysis consumes minimum time and resources as it does not involve the execution of the application by comparing to other analysis techniques for Android malware detection.[28-30] This technique is carried out by removing the features without implementing the application on a device or emulator. However, this technique has a significant downside of code obfuscation and dynamic code loading. Code obfuscation makes the pattern matching incapable of detecting inconsistency. On the other side, the advantages of static analysis is that it can detect possible security violations, runtime errors, and logical inconsistencies.[31-33] The permissions of API calls are usually used as static features as shown in Figure 1. The two main approaches for static analysis are *signature based* and *permission based*.

- *Signature-based approach*

This technique is also known as misuse detection technique. The commercial anti-malware yields are mostly used for the signature-based malware detection approaches. This technique provides a unique signature and removes the semantic patterns. Signature-based recognition is very capable for the already known malware, but the major problem of this approach is that it cannot identify the unknown malware types. Although the maximum malware remains undetected

TABLE 1 Comparison of static analysis and dynamic analysis

Components	Static analysis	Dynamic analysis
Target code execution	Not possible	Possible
Accuracy and effectiveness	Minimum as compared to dynamic analysis	Improved than static analysis
Time required	Minimum	Maximum
Benefits	Minimum cost and Minimum time required	Gives a profound examination and higher discovery rate with obscure malware location
Disadvantages	Constrained signature database and can identify only within the scope of known malware types	Power consumption and maximum time
Code obfuscation	Yes	No
Input	Binary files, scripting language files, etc	Memory snapshots, runtime API data

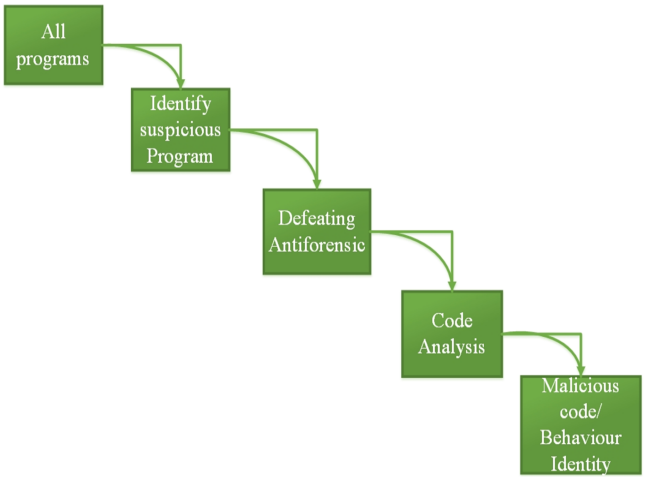

FIGURE 1 Static analysis process

due to the incomplete signature database, while we have detected the malware, its variants is required to be instantly update.[34,35]

- *Permission-based analysis*

Permission-based analysis is required where permission requested by an application shows a significant role in the access right. Applications must state which permissions they request or require in their manifest file. Android permission controls the required access to the application data. The data stored in the device cannot be accessed without the required permissions. AndroidManifest.XML file is present in the root directory, which stores essential information related to application on the Android system.[36,37] The permission mechanism should be working effectively before allowing the application to get the required asset. In any case, consents for every single proclaimed permission is not required for the particular application. It just examines the show document and no different records.[38,39]

2.2 | Dynamic analysis

Dynamic analysis is the way of testing and evaluating the data in real time. The main aim of this analysis is to repeatedly examine the code offline and detecting the run-time error. This approach aims to evaluate the malware by executing the application.[40,41] By comparing with static analysis, this approach is more complicated as resource consumption is performed in real environment. The benefit of this method is that it loads target information to determine the application behavior during runtime. Sandbox, virtual machine, and other forms of operation environment are constructed by the dynamic behavioral detection method. To acquire the application behavior model, it stimulates the execution of the application.[42,43] There are two main approach for dynamic analysis: *anomaly based* and *taint based*.

- *Anomaly-based detection*

The performance of this approach depends on machine learning algorithm in order to detect the malicious behavior. The features from the existing malware are used to train the model that predicts unknown malware. To perform in-depth analysis of malware detection, it requires a lot of effort and resources. Applications (apps) installed are used for identifying the malicious behavior in the system. The major drawback of this process is that, if it invokes more system call, then it may even classify for the legitimate application.[44]

- *Taint analysis*

It checks the user's input and modify the variables accordingly. It is the most used method as it focuses only on those apps that are shared to get sensible information. Dynamic taint analysis is a scientific technique that is used for this approach followed by Taint Droid. This technique called taint, which marks the data of interest with an identifier. When the information is being used, the taint stays with this information.[45] For tracking Android, the Train Droid provides system-wide data stream. The multisourced sensitive data like microphone, camera, and GPS can also be tracked by this method. Sensitive data and information can automatically be tagged by Taint Droid. Taint Droid records the tagged data from the system over any channel.[46,47] The fundamental problem of this system is that it cannot track data that leaves and returns the channel.

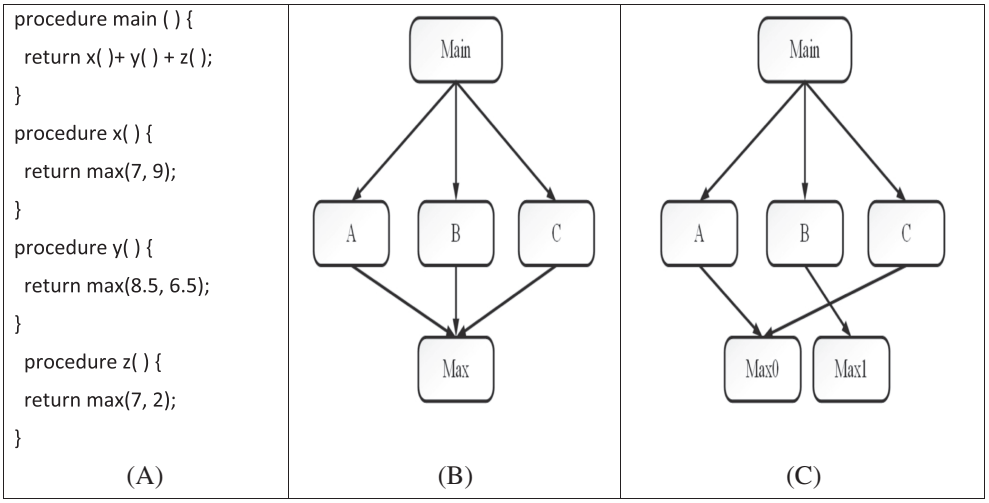

FIGURE 2 Call graphs for permission analysis ((A) example program, (B) context-insensitive, and (C) context-sensitive)[21]

2.3 | Call graph construction

Established algorithms neglect open-package assumption utilization situations, which is the reason that expansion code is not considered while separating remain private application. However, the expansion code can prompt direct call conditions between library strategies that are not obvious from the class chain.[48] The abbreviated call graphs for permission analysis is depicted in Figure 2. The UI string transitively calls toString() on every instance of painting JList. AbstractMap.toString() is called by giving the custom HashMap usage as substance, which does not supersede toString(). This method repeats over the passage set and calls getValue() on every section. The essential thing of the section set is the aggressor's link case. Like this, it effectively calls Expression.GetValue() that brilliantly summons System.SetSecurityManager (null). To methodically discover exploitable callback executions, a static exam should watch that there is no assailant callable strategy present, which transitively calls sensitive support without appropriate cleaning or permission checks. In any case, the static examination desires to do furthermore not forget that calls to callbacks are made plans to all workable put stock in executions. Cutting aspect name-chart calculations exclude a call part from name locales of Entry.GetValue() to the method Expression.GetValue(). However, this part is needed to discover the attack, which should be exhibited in right ways. If this edge is incorporated, information stream examination searching for unguarded ways to delicate activities are empowered to distinguish the vulnerability.[49]

3 | PERMISSION IMPLEMENTATION IN ANDROID APPLICATIONS

In the Android software program stack in which the Android framework degree, within the Android framework stage, a large portion of the contents are authorized. Using supplementary series ID, some of the permissions are implemented at the kernel level. If the application is the part of a group to acquire all of the privileges of that particular institution, it will be allowed to use supplementary organization ID.[50,51] Using the permissions declared in the programs that take place to file, the organizations are arranged. Figure 3 illustrates that the Bluetooth, digicam, and the Internet that are comparative examples of permissions, which can be enforced within the kernel level. The usage of group ID and other permissions such as SMS and MMS that are used in retrieving the communication list is enforced inside the Android framework stage.[52,53]

4 | ATTACK OF PERMISSION ESCALATION

The permission attacks are categorized and demonstrated in Figure 4. Confused deputy attack and collusion attack are the two types of permission escalation attack. Privileged benign applications in unprotected interfaces exploit the vulnerabilities by confused deputy attack.[13] To create a combined set of permissions, the collusion attack can be conceded out by multiple applications. These permit them to present malicious actions on an unauthorized performance.[14] The collusion attacks are ordered into two categories. They are direct collusion attack and indirect collusion attack. These two techniques are used for interconnecting with each other. The direct collusion way is used to communicate applications indirectly. The indirect collision attack is used where the application communicates with the third application.[54] Overt

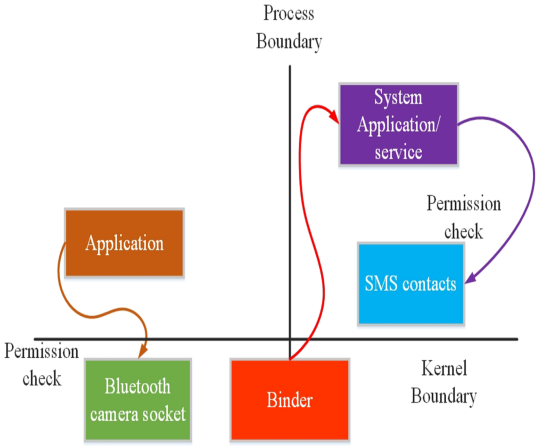

FIGURE 3 Permission enforcement in Android application

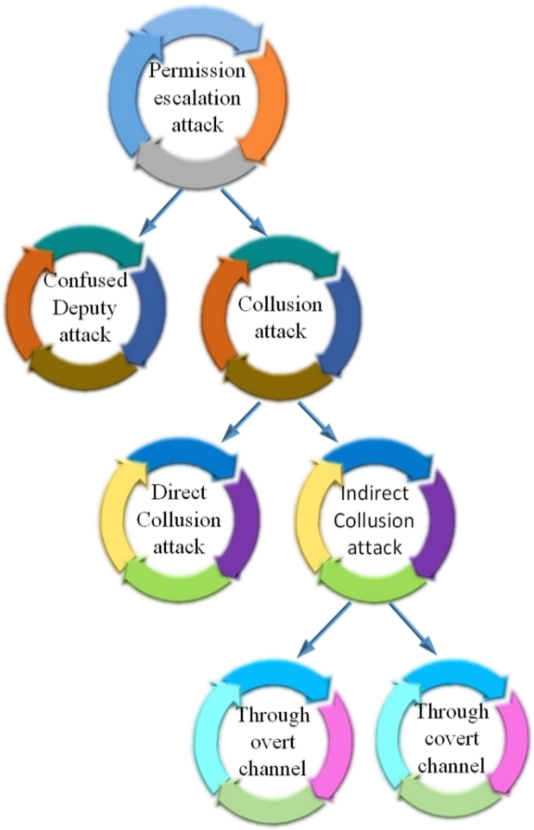

FIGURE 4 Permission escalation attack

channels or covert channels are provided by indirect collusion attack. The overt channels apply a data object to hold certain information that is like files, buffers, and input–output devices. Shared preferences, UNIX socket communication, and system logs are few other examples of overt channels.[55,56]

For communication, the covert channels are used objects, which are typically not intended. These objects are applied for covert channels with the Android's middleware layer bypass. At last, the measurement demonstrates that the low throughput of covert channels is appropriate to exchange private data.[38,57]

5 | MALICIOUS APPLICATION DETECTION USING ANDROID PERMISSIONS

5.1 | Static analysis–based approaches

Some major static analysis–based approaches from 2007 to 2019 are depicted in Table 2. In 2014, Fang et al[64] have investigated the various Android security issues caused by permission-based mechanisms. They have also reviewed the

TABLE 2 Analysis method based on static analysis

Analysis method	Research paper and year	Application
Static analysis method	Balzarotti et al[58] and 2007	Analysis of web-based applications
	Basin et al[59] and 2009	Automated analysis of security-design models
	Stolpe[60] and 2010	Permission based on the notion of derogation
	Jeon et al[61] and 2012	Fine-grained permissions in Android applications
	Zhou and Jiang[62] and 2012	To identify certainly malware applications
	Zhang et al[63] and 2013	Screening undesirable behaviors in android apps
	Fang et al[64] and 2014	To refer the concerns in android security
	Talha et al[65] and 2015	Static analysis to portray and characterize Android applications
	Song et al[66] and 2016	Incorporated static location and examination system for Android
	Rashidi et al[67] and 2016	Real-time expert recommendations
	Seshagiri et al[68] and 2016	Static code analysis of a framework capable of detecting the presence of malicious code
	Sokolova et al[69] and 2017	Android application classification and anomaly detection
	Li et al[70] and 2018	Significant Permission IDentification (SigPID), a malware detection system based on permission usage analysis to cope
	Liu et al[71] and 2019	Alde that influences the Xposed framework to accomplish analytics libraries in other apps

countermeasures of such issues based on their technical features. According to the authors, the android security can be enhanced by developing data driven methods strengthening the Android framework, flexible and fine-grained permission models irrespective of the current model that is based on coarse-grained and inflexible permission models, and maintaining the consistency between application intentions and system implementations.

Talha et al[65] have proposed a permission-based Android malware detection system called APK Auditor. The system uses static analysis technique for classifying the android applications into benign or malicious. The system consists of three parts, namely, a signature database, an android client, and a central server. The signature database stores the extracted data on applications as well as analytical results, whereas the android client is used by the end users to give application analysis request. The central server manages the whole analysis process by communicating with the database and the android client.

Song et al[66] have integrated the static detection method with the analysis framework where only the static detection method can result in high false rate and also the scope is limited. The proposed approach consists of four filtering layers, namely, the message digest values, combining malicious permissions, the dangerous permissions, and the dangerous intentions. The authors have proposed an intuitive threat-degree model for detecting the dangerous permissions.

Rashidi et al[67] have proposed RecDroid that creates a user-help-user environment for the android permission control. It is a crowdsourcing recommendation framework that aggregates the expert users' responses and then recommends it to the inexperienced users. Seshagiri et al[68] have proposed a static approach, namely, Amrita Malware Analyzer. This framework detects the malicious code by performing plaintext attack using strings contained in the malicious web pages.

Sokolova et al[69] have proposed a five-step methodology for finding the patterns of each category. The category patterns and key permissions are found using graph analysis metrics by modeling the required permissions as a graph. The authors

have proposed a privacy score based on the resemblance between an application and a given category pattern and a risk warning threshold based on the anticipated behavior of benign applications in distinct categories.

5.2 | Dynamic analysis–based approaches

Some major dynamic analysis–based approaches from 2007 to 2019 are depicted in Table 3. In 2013, Zhang et al[63] have proposed a dynamic analysis platform, namely, VetDroid. The VetDroid is used to construct permission use behaviors by identifying the explicit and implicit permission use points through correct permission information. It can also help in detecting data leaks, analyzing fine-grained causes of data leaks, and detecting susceptibility in regular applications.

Min and Cao[76] have proposed a runtime-based behavior analysis system for detecting the android malware. According to the authors, the traditional method, ie, signature-based method is not good enough for malware detection. The authors have analyzed 350 applications from third party Android market with good result.

Petsas et al[78] have investigated the anti-analysis techniques that can be used by malware to escape from dynamic analysis approaches. These techniques are based on three different categories, viz, static properties, dynamic heuristics, and VM-related complexities of Android emulator. Further, they proposed some countermeasures like modifying emulator, accurate binary translation, and hardware-assisted virtualization for improving dynamic analysis resistance against VM detection evasion.

Abah et al[79] have proposed a device monitoring system for an unrooted device. The system is used for collecting application data that is then used to feature vectors. These feature vectors describe the behavior of application for detecting the malware. Ab Razak et al[80] have developed behavior-based anomaly detection system to detect the deviation in application's network behavior. The system performs the traffic analysis for monitoring the suspicious network activities. The semisupervised machine leaning techniques are used for learning the normal behavioral patterns of the application. The system is used to detect mobile malware that cannot be identified by signature approach or by static or dynamic analysis method.

Thanigaivelan et al[82] have proposed a context-based dynamically reconfigurable access control system (CoDRA). The CoDRA uses feature-based policies that controls resource access and policy granularity. The policy enforcement was

TABLE 3 Analysis method based on dynamic analysis

Analysis method	Research Paper and year	Application
Dynamic analysis method	Centonze et al[72] and 2007	Automatic identification of precise access-control policies
	Zand and Ahmadian[73] and 2009	Application of homotopy analysis
	Blaschke[74] and 2010	Image analysis for remote sensing
	Isohara et al[75] and 2011	Android malware detection based on kernel
	Min and Cao[76] and 2012	Malware detection in Android by runtime-based behavior analysis of dynamic
	Amos et al[77] and 2013	Android malware identification by applying machine learning classifiers
	Petsas et al[78] and 2014	Hinder dynamic analysis of Android malware
	Abah et al[79] and 2015	Anomaly-based malware detection
	Ab Razak et al[80] And 2016	Application network behavior by web application
	Idrees et al[81] and 2017	Permission- and intent-based framework for identifying Android malware apps
	Thanigaivelan et al[82] and 2018	Context-based dynamically reconfigurable access control system for Android
	Ahmad et al[83] and 2019	StaDART analyzes the arguments of reflection APIs without framework modification

implemented by combining application behavior and resource features. The CoDRA uses static as well as dynamic constraints unlike the traditional that uses only static constraints on application activities.

6 | METHODOLOGY FOR THE LITERATURE REVIEW

Figure 5 depicts the overall process of literature review called SLR (systematic literature review). The literature review is done in the following steps.

- At the initial step, we characterize the exploration questions covered in the SLR and recognize the pertinent data from the publication in writing.
- In the next step, we enlist the distinctive search keywords that enable us to find the most prominent journals inside the extent of this SLR.
- The searching system itself is driven by two factors, the first being the striking generation storage facilities and the second highlights the approaches of scatterings from top scenes, which also included two diaries and meeting papers. Exclusion criteria are applied on the listed items to confine our analyses to exceptionally relevant papers, in this way sifting through papers of likely restricted intrigue.
- After that, we combine results obtained from both the tests as a list of publications to survey. Finally, future research is discussed.

6.1 | Research questions

RQ1. How to characterize the significant hazard levels for permissions while considering the recurrence of an event in malicious and typical applications?

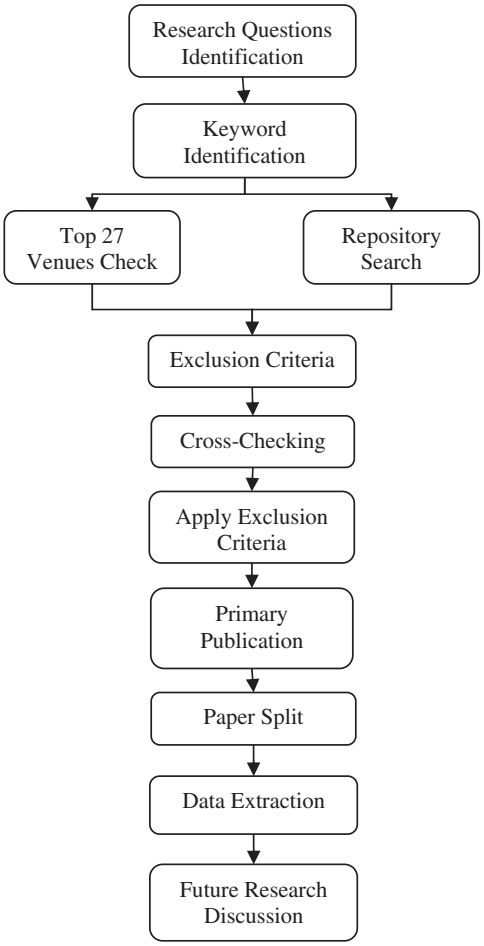

FIGURE 5 An overall diagram of systematic literature review

Few Android consents have been formally given by Google. To characterize, a risk level is an important part of the Android permission. The risk level depends on the event of permission in the malware and programming sets. To recognize malicious activities, the hazard level should be considered to be.

RQ2. How can one profile Android applications be utilizing dangers provided by consents on resources?

Android incorporates the likelihood to gather consents as indicated by assets to get to and to specific destinations. To capture messages being received by the client or to send messages for the benefit of the client the permission that enables the messages to the client. This gathering also contains permissions, which can be utilized to profit without their immediate association. In any case, amid the establishment, it is just the gathering, which is shown to the client, which implies that the client awards consents, for example, READ_SMS and others that are obscure.

RQ3. How machine learning and Android profiles help in malware detection?

Machine learning–based frameworks using Android profiles help in designing Android tests that categorize the rules as typical or malicious.

RQ4. How will security be concerned?

Two security concerns have been featured during investigation. We begin with some broad measurements and continue for possible smaller parts. Additionally, security issues have been portrayed in detail along with possible solutions.

6.2 | Search strategy

We have used search keywords and data sets as search strategy for finding the relevant publications.

1) Search keywords

The keywords associated with analysis activities, key aspects of permission analysis, and key aspects of static and dynamic analysis are the search terms as listed in Table 4, which are used in our review.

2) Search data sets

Data search depends on storehouses and, furthermore, is supplemented by check beside top venues in security. For finding the significant productions, repository search is proposed and the best scene check is utilized just as an extra checking process.

3) Repository search

Science Direct, Springer Link, IEEE Xplore Digital Library Web of Knowledge, Wiley Online Library, ACM Digital Library, Taylor and Francis, and Inderscience are the well-known electronic repositories to find the data sets of publications in the first attempt. Since now and again repository, search engine forces a point of confinement to measure of hunt result meta-information that one can download. The pursuit string is taken into consideration and emphasizing the point when we gather all pertinent meta-information of distributions. Science Direct permits to gather data on the first thousand things from its query items. Tragically, through applying our predefined search string, we get more than 10 000 outcomes on this repository.

6.3 | Top venue

We take into account all applications to ensure the repository search items considered in this paper. We have taken the main 27 venues for the SLR where 19 venues are from the software engineering and computer applications while the other eight venues are from the security and privacy field. Table 5 demonstrates these venues. The papers are taken from the IEEE Xplore, ACM, and Elsevier. In reality, the venues are not concentrated on permission analysis of the Android

TABLE 4 Search keywords

Line	Keywords
1.	Android permission analysis, permission in Android application
2.	Permission; authorization;
3.	Static*; dynamic*
4.	Android; Mobile; Smartphone*;
5.	Security; Privacy

TABLE 5 The top 27 venues containing both conference proceedings and journals in SE/CA and S&P fields

Acronym	*Full name*	*H- index*
	Software engineering and computer applications (SE/CA)	
JNCA	"Journal of Network and Computer Applications"	59
IJCIP	"International Journal of Critical Infrastructure Protection"	21
JERP	"Journal of Engineering Research and Applications"	21
JES	"Journal of Environmental Sciences"	65
IJCIS	"International Journal of Critical Infrastructure Protection"	21
ADVC	"Advances in Computers"	56
CEE	"Computers and Electrical Engineering"	40
FGCS	"Future Generation Computer Systems"	80
CN	"Computer Networks"	54
DI	"Digital Investigation"	37
ICC	"International Conference on Communications"	7
MobiSys	"International conference on Mobile systems"	14
IWCMC	"International Wireless Communications and Mobile Computing Conference"	7
TWC	"International Conference on Trust and trustworthy computing"	36
MONET	"Mobile Networks and Applications"	73
Pervasive Mobcomput	"Pervasive and Mobile Computing"	46
IWCMC	"International Wireless Communications and Mobile Computing Conference"	14
GJCST	"Global Journal of Computer Science and Technology: Network"	26
JCST	"Asian Journal of Computer Science and Technology"	5
	Security and privacy (S&P)	
COMPSEC	"Computers and Security"	40
CCS	"Computer and communications security"	71
CLSR	"Computer Law and Security Review"	19
ISI	"International Conference on Intelligence and Security Informatics"	6
NDSS	"Network and Distributed System Security Symposium"	56
IFS	"Information Forensics and Security"	77
SSP	"Symposium on Security and Privacy"	59
CIS	"Computational Intelligence and Security"	37

apps. The papers are collected from the years 2007 to 2019. The H-index mentioned in Table 5 is characterized by Google Scholar.

7 | EXCLUSION CRITERIA

The enquiry terms given above is extensive enough to collect a near about papers that matched title. At some point, this broadness might output improper results. For our SLR, we utilize the avoidance criteria.

- To start, we sift through non-English composed journals as English is the typical language used by the analysts, and most of the logical works are covered by it.
- Secondly, our focus is on broad works with detailed productions. In this manner, we reject short papers with fewer than seven pages in LNCS single column. The making of these papers is a regular process and these need to update to the previous papers.
- In the third step, the copied papers are identified. The papers are checked firstly by looking at the sequence of the authors, writing skill, and paper titles. The similarity ratio of the suspected papers is checked. At this point, when copied data is affirmed, we sift through the less broad publication.
- In the fourth step, we incorporate "mobile" term for gathering most papers; the gathered lot papers include about "portable systems administration" or iOS or Windows stages. We avoid this kind of non-Android papers. In this way, a portion of the gathered papers are rejected that helps in manually evaluating and collecting the outstanding papers.
- In the fifth step, we rapidly go through the rest of the papers and only accept those that are related to Android, however, does not hold permission analyzing procedures. For instance, publications of static analysis of Android apps are barred.

- We avoid manuscripts that statically describe Android OS rather than Android apps as our focus is to review works related to permission analysis of Android apps. For example, we have expelled EdgeMiner[84] and PSCout,[52] these paper are related to Android structure.
- We reject manuscripts that do not explain the app program code. For example, papers that perform a static investigation on the meta-information of applications are barred.
- We sift through specialized reports like SCanDroid.[85] Such non–peer-reviewed papers are frequently redistributed in a gathering and likely to be incorporated into our inquiry set with an alternate title. For example, the specialized paper on IccTA[86] has showed up in collected papers.
- We additionally exclude papers that use the announcement groupings or grep API names from the source/application code. For instance, we avoid Juxtapp,[87] a clone identification approach, from thought since it just takes opcode groupings for its permission analysis.

8 | EXTRACTION OF DATA

Material papers must be isolated from each paper once the papers have been assembled in order to produce a logical classification of the information. The material papers are isolated to cover up the investigation questions recorded beforehand. This technique helps to gather each paper's assessment and give an outline of data separated from the major publishers. Figure 6 shows the method used in separating data from the chosen publications.

Directed problems: Methodologies are additionally ordered on the directed issues. Cases of issues incorporate protection spillage, permission administration, energy enhancement, and so on.

Fundamental techniques: This measurement concentrates on the key systems embraced by analyzed essential productions. The central strategies in this work incorporate methods like program analysis that take care of issues through various means like WALA or Soot.

Features of permission analysis: This measurement incorporates permission analysis of relevant highlights. It is utilized to confirm whether it is path sensitive, alias-aware, static-aware context-sensitive, flow-sensitive, implicit flow-aware, field-sensitive, object-sensitive for an essential product.

Android characteristics: This measurement incorporates such attributes that are firmly identified with Android, for example, IAC (Inter-App Communication), ICC, and framework.

Evaluation strategies: This measurement concentrates on the assessment techniques for required distributions and used to response to the inquiry of how their methodologies are assessed. Finally, this measurement will check whether their methodologies are assessed over in artificial apps and real-world apps.

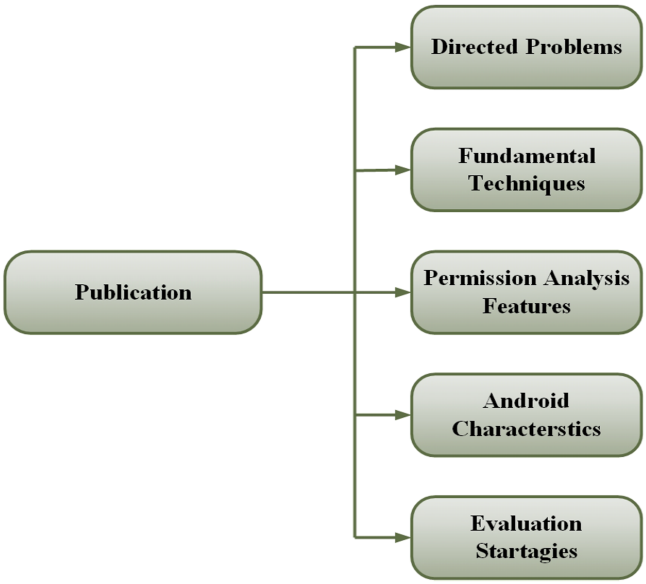

FIGURE 6 Outline of data separated from the major publishers

9 | FUTURE RESEARCH DISCUSSION

Android malware analysis is slanting the correct way. Various antivirus products and essential solutions do give insurance opposed to the majority of malware; yet, techniques, for example, bytecode marks, are frail opposed to the developing contemporary malware. We, in this way, propose the accompanying areas for future research.

9.1 | Multilevel analysis and hybrid analysis

First, static solutions are started to solidify against inconsequential confusions; however, numerous applications and most malware areas are now utilizing more significant amounts of obfuscation.[88] As recent static frameworks are yet viable and versatile, we propose that in the situations where obfuscation is distinguished and dynamic analysis can be utilized as a part of conjunction with finishing. Then again, dynamic solutions intrinsically have insignificant code scope, however, utilize static study to direct analysis over more ways[89] or utilize applications exercisers like Monkey Runner, manual info, or occasion injectors.[90]

Therefore, hybrid solutions could consolidate dynamic and static analysis in such ways that their additional qualities alleviate the shortcoming of each of them. Harvester[91] device can diminish confusion created by encoded strings and intelligent techniques with its hybrid strategies. It additionally appears to be useful to create multilevel frameworks, as it regularly gives more, and wealthier, highlights.[92] On top of that, if a multilevel analysis is conducted, then the malware will not be able to hide. Multilevel analyses are upgraded by parallel handling and give quicker identification frameworks.[93] The drawback of this multilevel technique, in any case, is that it can cause extensive extra overhead, diminish straightforwardness, increment odds of code bugs, and might be less convenient.[94]

9.2 | Code coverage

Covering the code is essential for entire, robust malware analyses. In accordance with the stats, this is troublesome while overseeing logically stacked code, neighbourhood code, and framework-based activity.[95] Powerfully, this is invigorating, as only a solitary way is exhibited per execution, customer affiliations are difficult to mechanize, and also malware may have part practices. There are a couple of points of interest to dynamic out-of-the-case arrangements, taking in account the dispatch of ART,[96] for example, having the ability to adjust to different available Android frames and to bar malware maintaining a strategic distance from examinations with neighbourhood code or reflection. For example, framework get driven examination is out-of-the-case, yet can, at present, break down Android-level practices and dynamic framework and can usefully handle root problems.[97] While half-and-half arrangements and cannier inductions (eg, IntelliDroid that is a static and dynamic API-based information generator) would gigantically expand code scope, unmistakable systems ought to be also inquired about into in perspective of malware designs. As an example, we can take that, while manual information is ordinarily not flexible, crowdsourcing[98] may be a fascinating system. Regardless, zero-day slow malware will display troubles as time is relied upon to accumulate customer input.[99]

Code coverage furthermore displays a captivating request on whether malware tends to use "viable" picked approaches to execute more noxious conduct or more laborious approaches to dodge location.[100] It would help perceive malware designs and augment the ampleness of future examinations in future research.[101] Another subject that should be significant is recognizing and understanding sets of malware to see which permission(s) conceivably trigger(s), for example, explicit consents or triggers like customer UI or structure events.[102] Moreover, applications should be continued running on a couple of assorted Android OS shapes as different adjustments have unmistakable plans of vulnerabilities by manufacturing the code scope. To the Dalvik VM, this would be extensively harder to execute or the OS to suit for irregular state examinations yet functional without-of-the-container investigations.[103]

9.3 | Virtualization and hybrid devices

Other than smart stimuli, adjusting emulators for extended straightforwardness, eg, sensible phone identifiers or using emulators with access to bonafide physical hardware like sensors, functional GPS, an accelerometer to trap VM-mindful malware, may demonstrate valuable and intriguing.[104] More current, more advanced, malware from 2014 and 2015 are ending up progressively. However, accomplishing an impeccable emulator is, tragically, unfeasible. A planning assault, where specific operations are coordinated for disparities, is yet open issues for conventional malware and is hard to trick.[105]

Besides, malware, for example, Android.HeHe and DenDroid cannot be identified but still can cause great harm. Based on a past report, malware can alert of a few gadget highlights to distinguish emulators.[106] This incorporates, yet does not stop at, the gadget IMEI, directing table, timing assaults, reasonable sensory yield, and the serial number of the gadget.[107] It is likewise conceivable to unique finger impression and recognizes specific imitated situations, for example, extraordinary dynamic analysis structures, through the gadget execution highlights previously mentioned.[108] One solution for this issue is utilizing genuine gadgets in every powerful test. Be that as it may, this makes analyzing vast malware sets a costly and challenging undertaking, and the same number of gadgets would be required and additionally an approach to re-establish a gadget to a perfect condition for fast, proficient, and reliable analysis.[109,110]

10 | CONCLUSION

Permission-based analysis and an integral feature of Android malware analysis are presented in this paper. Permission analysis is a progressing approach that uncovers the major security issues and rapidly developing application code statically. For this review, 110 research articles are gathered, which are already published in security and privacy journal and conferences for different programming languages with software engineering methodology. The Android permission protocol is having the severe security implications because the real-world Android application study confirms the findings but the Android permission protocol has several flaw. The various cases permit the attacker to evade the permission checks entirely. Although, we have covered application permission–based malware but a new type of malware family is developing as android market is growing very fast. A large-scale study on the above topic might give more clarity of the subject. It is also required to study a category that includes users too so that permissions can categorize more precisely.

ACKNOWLEDGMENTS

This study is a part of the Ph.D. Work of the author in Department of CSE, National Institute of Technology Patna, India and supported by the National Institute of Technology Patna, India, and partially supported by the National Funding from the Fundação para a Ciência e a Tecnologia (FCT) through the UID/EEA/50008/2019 Project; and by the Brazilian National Council for Research and Development (CNPq) under grant 309335/2017-5.

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

ORCID

Gulshan Shrivastava https://orcid.org/0000-0003-3671-4921
Joel J. P. C. Rodrigues https://orcid.org/0000-0001-8657-3800

REFERENCES

1. Kaur S, Kaur M. Review paper on implementing security on Android application. *J Environ Sci Comput Sci Eng Technol*. 2013;2(3).
2. Powar S, Meshram BB. Survey on Android security framework. *Int J Eng Res Appl*. 2013;3(2):907-911.
3. Felt AP, Chin E, Hanna S, Song D, Wagner D. Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11); 2011; Chicago, IL.
4. Moore SR, Ge H, Li N, Proctor RW. Cybersecurity for Android applications: permissions in Android 5 and 6. *Int J Hum Comput Interact*. 2019;35(7):630-640.
5. Garg S, Baliyan N. A novel parallel classifier scheme for vulnerability detection in Android. *Comput Electr Eng*. 2019;77:12-26.
6. Mollah MB, Azad AK, Vasilakos A. Security and privacy challenges in mobile cloud computing: survey and way ahead. *J Netw Comput Appl*. 2017;84:38-54.
7. Ntantogian C, Apostolopoulos D, Marinakis G, Xenakis C. Evaluating the privacy of Android mobile applications under forensic analysis. *Comput Secur*. 2014;42:66-76.
8. Pennekamp J, Henze M, Wehrle K. A survey on the evolution of privacy enforcement on smartphones and the road ahead. *Pervasive Mob Comput*. 2017;42:58-76.
9. Brandom R. https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote. Accessed August 2019.
10. Raj M, Di Francesco M, Das SK. Secure mobile cloud computing. In: *Handbook on Securing Cyber-Physical Critical Infrastructure*. Amsterdam, Netherlands: Elsevier; 2012:411.

11. Amalfitano D, Fasolino AR, Tramontana P, Robbins B. Testing Android mobile applications: challenges, strategies, and approaches. In: *Advances in Computers*; vol. 89. Amsterdam, Netherlands: Elsevier; 2013:1-52.
12. Fernando N, Loke SW, Rahayu W. Mobile cloud computing: a survey. *Future Gener Comput Syst*. 2013;29(1):84-106.
13. Girolami M, Chessa S, Caruso A. On service discovery in mobile social networks: survey and perspectives. *Computer Networks*. 2015;88:51-71.
14. Clarke R. The prospects of easier security for small organisations and consumers. *Comput Law Secur Rev*. 2015;31(4):538-552.
15. Armando A, Merlo A, Verderame L. Security considerations related to the use of mobile devices in the operation of critical infrastructures. *Int J Crit Infrastructure Prot*. 2014;7(4):247-256.
16. Korkmaz I, Metin SK, Gurek A, Gur C, Gurakin C, Akdeniz M. A cloud based and Android supported scalable home automation system. *Comput Electr Eng*. 2015;43:112-128.
17. Virvilis N, Mylonas A, Tsalis N, Gritzalis D. Security Busters: web browser security vs. rogue sites. *Comput Secur*. 2015;52:90-105.
18. Shrivastava G, Kumar P. SensDroid: analysis for malicious activity risk of Android application. In: *Multimedia Tools and Applications*. Berlin, Germany: Springer; 2019. https://doi.org/10.1007/s11042-019-07899-1
19. Chin E, Felt AP, Greenwood K, Wagner D. Analyzing inter-application communication in Android. In: Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11); 2011; Bethesda, MD.
20. Singh V, Sharma K. Smartphone security: review of challenges and solution. In: Proceedings of the 2nd International Conference on Information and Communication Technology for Competitive Strategies; 2016; Udaipur, India.
21. Liu Y, Wang SC, Yang Y, Chen YC, Sun HM. An automatic UI interaction script generator for Android applications using activity call graph analysis. *EURASIA J Math Sci Technol Educ*. 2018;14(7):3159-3179.
22. He Y, Yang X, Hu B, Wang W. Dynamic privacy leakage analysis of Android third-party libraries. *J Inf Secur Appl*. 2019;46:259-270.
23. Zhang J, Qin Z, Zhang K, Yin H, Zou J. Dalvik opcode graph based Android malware variants detection using global topology features. *IEEE Access*. 2018;6:51964-51974.
24. Zhou Y, Zhang X, Jiang X, Freeh VW. Taming information-stealing smartphone applications (on Android). In: *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011*. Berlin, Germany: Springer; 2011:93-107.
25. Shrivastava G, Kumar P. Privacy analysis of Android applications: state-of-art and literary assessment. *Scalable Comput Pract Exp*. 2017;18(3):243-252.
26. Enck W, Octeau D, McDaniel P, Chaudhuri S. A study of Android application security. In: Proceedings of the USENIX Security Symposium; 2011; San Francisco, CA.
27. Faruki P, Ganmoor V, Laxmi V, Gaur MS, Bharmal A. AndroSimilar: robust statistical feature signature for Android malware detection. In: Proceedings of the 6th International Conference on Security of Information and Networks (SIN '13); 2013; Aksaray, Turkey.
28. Omar M, Mohammed D, Nguyen V, Dawson M, Banisakher M. Android application security. In: *Applying Methods of Scientific Inquiry Into Intelligence, Security, and Counterterrorism*. Hershey, PA: IGI Global; 2019:46-67.
29. Zhang Y, Yang M, Yang Z, Gu G, Ning P, Zang B. Permission use analysis for vetting undesirable behaviors in Android apps. *IEEE Trans Inf Forensics Secur*. 2014;9(11):1828-1842.
30. Sharma K, Gupta BB. Attack in smartphone Wi-Fi access channel: state of the art, current issues, and challenges. In: *Next-Generation Networks*. Singapore: Springer; 2018:555-561.
31. Cheng X, Fang L, Hong X, Yang L. Exploiting mobile big data: sources, features, and applications. *IEEE Network*. 2017;31(1):72-79.
32. Moser A, Kruegel C, Kirda E. Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007); 2007; Miami Beach, FL.
33. Batyuk L, Herpich M, Camtepe SA, Raddatz K, Schmidt A-D, Albayrak S. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. In: Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software; 2011; Fajardo, Puerto Rico.
34. Xu J, Yu Y, Chen Z, et al. MobSafe: cloud computing based forensic analysis for massive mobile applications using data mining. *Tsinghua Sci Technol*. 2013;18(4):418-427.
35. Xue Y, Tan YA, Liang C, Li Y, Zheng J, Zhang Q. RootAgency: a digital signature-based root privilege management agency for cloud terminal devices. *Information Sciences*. 2018;444:36-50.
36. Sharma K, Gupta BB. Towards privacy risk analysis in Android applications using machine learning approaches. *Int J E Serv Mob Appl*. 2019;11(2):1-21.
37. Mustafa T, Sohr K. Understanding the implemented access control policy of Android system services with slicing and extended static checking. *Int J Inf Secur*. 2015;14(4):347-366.
38. Raveendranath R, Rajamani V, Babu AJ, Datta SK. Android malware attacks and countermeasures: current and future directions. In: Proceedings of the 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT); 2014; Kanyakumari, India.
39. Wang W, Wang X, Feng D, Liu J, Han Z, Zhang X. Exploring permission-induced risk in Android applications for malicious application detection. *IEEE Trans Inf Forensics Secur*. 2014;9(11):1869-1882.
40. Yang Z, Yang M. Leakminer: detect information leakage on Android with static taint analysis. In: Proceedings of the 2012 3rd World Congress on Software Engineering; 2012; Wuhan, China.
41. Homayoun S, Ahmadzadeh M, Hashemi S, Dehghantanha A, Khayami R. BoTShark: a deep learning approach for botnet traffic detection. In: *Cyber Threat Intelligence*. Cham, Switzerland: Springer; 2018:137-153.

42. Dietz M, Shekhar S, Pisetsky Y, Shu A, Wallach DS. Quire: lightweight provenance for smart phone operating systems. In: Proceedings of the USENIX Security Symposium; 2011; San Francisco, CA.
43. Milosevic N, Dehghantanha A, Choo K-KR. Machine learning aided Android malware classification. *Comput Electr Eng*. 2017;61:266-274.
44. Schlegel R, Zhang K, Zhou XY, Intwala M, Kapadia A, Wang X. Soundcomber: a stealthy and context-aware sound Trojan for smartphones. In: Proceedings of the Network and Distributed System Security Symposium (NDSS 2011); 2011; San Diego, CA.
45. Nauman M, Khan S, Zhang X. Apex: extending Android permission model and enforcement with user-defined runtime constraints. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security; 2010; Beijing, China.
46. Zhang J, Tian C, Duan Z. FastDroid: efficient taint analysis for Android applications. In: Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19); 2019; Montreal, Canada.
47. Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi A-R, Shastry B. Poster: the quest for security against privilege escalation attacks on Android. In: Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11); 2011; Chicago, IL.
48. Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi A-R. *Xmandroid: A New Android Evolution to Mitigate Privilege Escalation Attacks*. Technical Report TR-2011-04. Darmstadt, Germany: Technische Universität Darmstadt; 2011.
49. Hinarejos MF, Almenárez F, Cabarcos PA, Ferrer-Gomila JL, López AM. Risklaine: a probabilistic approach for assessing risk in certificate-based security. *IEEE Trans Inf Forensics Secur*. 2018;13(8):1975-1988.
50. Fan M, Liu J, Wang W, Li H, Tian Z, Liu T. DAPASA: detecting Android piggybacked apps through sensitive subgraph analysis. *IEEE Trans Inf Forensics Secur*. 2017;12(8):1772-1785.
51. Schmidt A-D, Bye R, Schmidt H-G, et al. Static analysis of executables for collaborative malware detection on Android. In: Proceedings of the 2009 IEEE International Conference on Communications; 2009; Dresden, Germany.
52. Au KWY, Zhou YF, Huang Z, Lie D. Pscout: analyzing the Android permission specification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12); 2012; Raleigh, NC.
53. Meier R. *Professional Android 4 Application Development*. Indianapolis, IN: John Wiley & Sons; 2012.
54. Buchinger S, Kriglstein S, Brandt S, Hlavacs H. A survey on user studies and technical aspects of mobile multimedia applications. *Entertainment Computing*. 2011;2(3):175-190.
55. Mislan RP, Casey E, Kessler GC. The growing need for on-scene triage of mobile devices. *Digital Investigation*. 2010;6(3-4):112-124.
56. Xue W-F, Homans SW, Radford SE. Systematic analysis of nucleation-dependent polymerization reveals new insights into the mechanism of amyloid self-assembly. *Proc Natl Acad Sci*. 2008;105(26):8926-8931.
57. Shrivastava G, Kumar P. Android application behavioural analysis for data leakage. *Expert Systems*. 2019. https://doi.org/10.1111/exsy.12468
58. Balzarotti D, Cova M, Felmetsger VV, Vigna G. Multi-module vulnerability analysis of web-based applications. In: Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07); 2007; Alexandria, VA.
59. Basin D, Clavel M, Doser J, Egea M. Automated analysis of security-design models. *Inf Softw Technol*. 2009;51(5):815-831.
60. Stolpe A. A theory of permission based on the notion of derogation. *J Appl Log*. 2010;8(1):97-113.
61. Jeon J, Micinski KK, Vaughan JA, et al. Dr. Android and Mr. Hide: fine-grained permissions in Android applications. In: Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices; 2012; Raleigh, NC.
62. Zhou Y, Jiang X. Dissecting Android malware: characterization and evolution. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy; 2012; San Francisco, CA.
63. Zhang Y, Yang M, Xu B, et al. Vetting undesirable behaviors in Android apps with permission use analysis. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13); 2013; Berlin, Germany.
64. Fang Z, Han W, Li Y. Permission based Android security: issues and countermeasures. *Comput Secur*. 2014;43:205-218.
65. Talha KA, Alper DI, Aydin C. APK Auditor: permission-based Android malware detection system. *Digital Investigation*. 2015;13:1-4.
66. Song J, Han C, Wang K, Zhao J, Ranjan R, Wang L. An integrated static detection and analysis framework for Android. *Pervasive Mob Comput*. 2016;32:15-25.
67. Rashidi B, Fung C, Vu T. Android fine-grained permission control system with real-time expert recommendations. *Pervasive Mob Comput*. 2016;32:62-77.
68. Seshagiri P, Vazhayil A, Sriram P. AMA: static code analysis of web page for the detection of malicious scripts. *Procedia Comput Sci*. 2016;93:768-773.
69. Sokolova K, Perez C, Lemercier M. Android application classification and anomaly detection with graph-based permission patterns. *Decis Support Syst*. 2017;93:62-76.
70. Li J, Sun L, Yan Q, Li Z, Srisa-an W, Ye H. Significant permission identification for machine-learning-based Android malware detection. *IEEE Trans Ind Inform*. 2018;14(7):3216-3225.
71. Liu X, Liu J, Zhu S, Wang W, Zhang X. Privacy risk analysis and mitigation of analytics libraries in the Android ecosystem. *IEEE Trans Mob Comput*. 2019.
72. Centonze P, Flynn RJ, Pistoia M. Combining static and dynamic analysis for automatic identification of precise access-control policies. In: Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007); 2007; Miami Beach, FL.
73. Zand MM, Ahmadian MT. Application of homotopy analysis method in studying dynamic pull-in instability of microsystems. *Mech Res Commun*. 2009;36(7):851-858.
74. Blaschke T. Object based image analysis for remote sensing. *ISPRS J Photogramm Remote Sens*. 2010;65(1):2-16.
75. Isohara T, Takemori K, Kubota A. Kernel-based behavior analysis for Android malware detection. In: Proceedings of the 2011 7th International Conference on Computational Intelligence and Security; 2011; Hainan, China.

76. Min LX, Cao QH. Runtime-based behavior dynamic analysis system for Android malware detection. *Adv Mater Res*. 2013;756-759:2220-2225.
77. Amos B, Turner H, White J. Applying machine learning classifiers to dynamic Android malware detection at scale. In: Proceedings of the 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC); 2013; Sardinia, Italy.
78. Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, Ioannidis S. Rage against the virtual machine: hindering dynamic analysis of Android malware. In: Proceedings of the 7th European Workshop on System Security (EuroSec '14); 2014; Amsterdam, The Netherlands.
79. Abah J, Waziri OV, Abdullahi MB, Ume UA, Adewale OS. Extracting Android applications data for anomaly-based malware detection. *Glob J Comput Sci Technol*. 2015;15.
80. Ab Razak MF, Anuar NB, Salleh R, Firdaus A. The rise of "malware": bibliometric analysis of malware study. *J Netw Comput Appl*. 2016;75:58-76.
81. Idrees F, Rajarajan M, Conti M, Chen TM, Rahulamathavan Y. PIndroid: a novel Android malware detection system using ensemble learning methods. *Comput Secur*. 2017;68:36-46.
82. Thanigaivelan NK, Nigussie E, Hakkala A, Virtanen S, Isoaho J. CoDRA: context-based dynamically reconfigurable access control system for Android. *J Netw Comput Appl*. 2018;101:1-7.
83. Ahmad M, Costamagna V, Crispo B, Bergadano F, Zhauniarovich Y. StaDART: addressing the problem of dynamic code updates in the security analysis of Android applications. *J Syst Softw*. 2019.
84. Cao Y, Fratantonio Y, Bianchi A, et al. EdgeMiner: automatically detecting implicit control flow transitions through the Android framework. In: Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS 2015); 2015; San Diego, CA.
85. Spreitzer R, Palfinger G, Mangard S. SCAnDroid: automated side-channel analysis of Android APIs. In: Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSec '18); 2018; Stockholm, Sweden.
86. Li L, Bartel A, Bissyandé TF, et al. IccTA: detecting inter-component privacy leaks in Android apps. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1 (ICSE '15); 2015; Florence, Italy.
87. Hanna S, Huang L, Wu E, Li S, Chen C, Song D. Juxtapp: a scalable system for detecting code reuse among Android applications. In: *Detection of Intrusions and Malware, and Vulnerability Assessment: 9th International Conference, DIMVA 2012, Heraklion, Crete, Greece, July 26-27, 2012, Revised Selected Papers*. Berlin, Germany: Springer; 2012:62-81.
88. Unuchek R, Chebyshev V. Mobile malware evolution 2015. AO Kapersky Lab. 2018.
89. Felt AP, Finifter M, Chin E, Hanna S, Wagner D. A survey of mobile malware in the wild. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11); 2011; Chicago, IL.
90. Feng Y, Anand S, Dillig I, Aiken A. Apposcopy: semantics-based detection of Android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering; 2014; Hong Kong, China.
91. Zheng C, Zhu S, Dai S, et al. Smartdroid: an automatic system for revealing ui-based trigger conditions in Android applications. In: Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices; 2012; Raleigh, NC.
92. Arzt S, Rasthofer S, Fritz C, et al. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. *ACM SIGPLAN Notices*. 2014;49(6):259-269.
93. Dini G, Martinelli F, Saracino A, Sgandurra D. MADAM: a multi-level anomaly detector for Android malware. In: *Computer Network Security: 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2012, St. Petersburg, Russia, October 17-19, 2012. Proceedings*. Berlin, Germany: Springer; 2012:240-253.
94. Mahmood R, Mirzaei N, Malek S. Evodroid: segmented evolutionary testing of Android apps. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014); 2014; Hong Kong, China.
95. Deng L, Offutt J, Ammann P, Mirzaei N. Mutation operators for testing Android apps. *Inf Softw Technol*. 2017;81:154-168.
96. Georgiev AB, Sillitti A, Succi G. Open source mobile virtual machines: an energy assessment of Dalvik vs. ART. In: *Open Source Software: Mobile Open Source Technologies: 10th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2014, San José, Costa Rica, May 6-9, 2014. Proceedings*. Berlin, Germany: Springer; 2014:93-102.
97. Tam K, Edwards N, Cavallaro L. Detecting Android malware using memory image forensics. In: Proceedings of the Engineering Secure Software and Systems (ESSoS) Doctoral Symposium; 2015; Milan, Italy.
98. Agarwal Y, Hall M. ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In: Proceedings of the 11th Annual International Conference on Mobile Systems, Applications, and Services; 2013; Taipei, Taiwan.
99. Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, Cavallaro L. DroidSieve: fast and accurate classification of obfuscated Android malware. In: Proceedings of the 7th ACM on Conference on Data and Application Security and Privacy (CODASPY '17); 2017; Scottsdale, AZ.
100. Bhandari S, Panihar R, Naval S, Laxmi V, Zemmari A, Gaur MS. SWORD: semantic aware Android malware detector. *J Inf Secur Appl*. 2018;42:46-56.
101. Afonso V, Bianchi A, Fratantonio Y, et al. Going native: using a large-scale analysis of Android apps to create a practical native-code sandboxing policy. In: Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS 2016); 2016; San Diego, CA.
102. Feizollah A, Anuar NB, Salleh R, Suarez-Tangil G, Furnell S. AndroDialysis: analysis of Android intent effectiveness in malware detection. *Comput Secur*. 2017;65:121-134.

103. Li L, Bissyandé TF, Octeau D, Klein J. Droidra: taming reflection to support whole-program analysis of Android apps. In: Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016); 2016; Saarbrücken, Germany.
104. Zaddach J, Bruno L, Francillon A, Balzarotti D. AVATAR: a framework to support dynamic security analysis of embedded systems' firmwares. In: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS 2014); 2014; San Diego, CA.
105. 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). Article No. 76.
106. Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Blasco J. Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families. *Expert Syst Appl*. 2014;41(4):1104-1117.
107. Sharma K, Gupta BB. Mitigation and risk factor analysis of Android applications. *Comput Electr Eng*. 2018;71:416-430.
108. Maier D, Müller T, Protsenko M. Divide-and-conquer: why Android malware cannot be stopped. In: Proceedings of the 2014 9th International Conference on Availability, Reliability and Security; 2014; Fribourg, Switzerland.
109. Wong MY, Lie D. IntelliDroid: a targeted input generator for the dynamic analysis of Android malware. In: Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS 2016); 2016; San Diego, CA.
110. Li L, Li D, Bissyandé TF, et al. Understanding Android app piggybacking: a systematic study of malicious code grafting. *IEEE Trans Inf Forensics Secur*. 2017;12(6):1269-1284.

How to cite this article: Shrivastava G, Kumar P, Gupta D, Rodrigues JJPC. Privacy issues of android application permissions: A literature review. *Trans Emerging Tel Tech*. 2019;e3773. https://doi.org/10.1002/ett.3773