User Privacy and Data Flow Control for Android Apps: Systematic Literature Review

Zainab Rashid Alkindi[1,*], Mohamed Sarrab[2] and Nasser Alzeidi[1]

[1]*Department of Computer Science, College of Science, Sultan Qaboos University, Muscat, Oman*
[2]*Communication & Information Research Center, Sultan Qaboos University, Muscat, Oman*
E-mail: s109871@student.squ.edu.om; sarrb@squ.edu.om; alzidi@squ.edu.om
[*]*Corresponding Author*

Received 21 November 2020; Accepted 01 February 2021;
Publication 15 March 2021

Abstract

Android mobile apps gain access to numerous users' private data. Users of different Android mobile apps have less control over their sensitive data during installation and run-time processes. Too often, these apps consider data privacy less serious than users' expectations. Many mobile apps misbehave and upload users' data without permission which confirmed the possibility of privacy leakage through different network channels. The literature has proposed various approaches to protect user's data and avoid privacy violations. This paper, provides a comprehensive overview of state-of-art research on Android user privacy, and data flow control. The aim is to highlight the main trends, pinpoint the main methodologies applied, and enumerate the privacy violations faced by Android users. We also shed some light on the directions where the researcher's community effort is still needed. To this end, we conduct a Systematic Literature Review (SLR) during which we surveyed 109 relevant research papers published in leading conferences and journals. Our thorough examination of the relevant literature has led to a critical analysis of the proposed solutions with a focus on user privacy

Journal of Cyber Security and Mobility, Vol. 10_1, 261–304.
doi: 10.13052/jcsm2245-1439.1019

extensions and mechanisms for the Android mobile platform. Furthermore, possible solutions and research directions have been discussed.

Keywords: User privacy, data flow control, Android apps, mobile application.

1 Introduction

Android mobile apps have capabilities to provide high demand services for customers and store a high level of user's data, i.e., geographical data, personal contacts, multimedia exchanges, commercial transactions, and social networking. Thus, users' private data required by several apps have violently and led to greater security and privacy concerns. Between December 2009 and August 2020, the number of available apps in the Google Play Store was most recently placed at 2.96 million apps. The Invisible Digital Threat: Mobile Ad Fraud 2019 Report published by Malkiel and Stoke [1], over 29,000 malicious apps on the Google platform were discovered during Q1 of 2020, in comparison to 14,500 during the same quarter of last year. The report stated that the top 100 most active malicious apps in 2019 that Secure-D blocked, under a third (32%) of them are still currently available to download on Google Play.

The Android platform adopts different privacy and security practices to protect sensitive properties from violation apps. Currently, the numbers of attackers and malicious have raised rapidly [2, 3]. Privacy concerns in the Android OS are growing, whereas many research studies and reports have been published, efforts have been carried out to detect and analyses privacy leakage either statically, or dynamically, or using hybrid techniques [4–8]. Furthermore, privacy infiltration is an information flow security problem where the user's confidential data leakage via social networks without users' prior knowledge or permission. Nevertheless, after permissions are approved by users at installation time, apps could use these authorizations to access sensitive data without any control or restrictions [3]. Therefore, securing such data privacy is observed as the most valuable and considerable discussion regarding issues, trustees, consistencies, and accuracy.

Many studies focused on solutions to construct effective behavior representations that are based on Android OS [2, 9–13]. The research was focused on providing users with a technique to check apps permissions. Besides, the Android market (Google Play) allows developers to publish their applications with normal review and delicate approval process. Currently, Google play

refers to mobile users' feedbacks to identify policy violations [14, 15]. Besides, [14] stated that around 49% of the Google play apps are violating ad policies by pushing notifications, adding home screen icons, and altering browser settings. Only 8% acting as malicious behavior, such as downloading evil files to users' mobile phones [14]. Hence, the users' data can be downloaded, uploaded, and shared over different mobile cloud services. The user privacy data have numerous performance-related impediments, e.g., bandwidth, storage size, and battery lifetime, as well as environment-related problems, e.g., availability, scalability, and heterogeneity [16]. Since most of the services accessed by users are from cloud and external hosts, it is worth addressing the security properties that must be included in a different mobile platform environment. There are several mobile attacks i.e. application-based attacks, web-based attacks, network-based attacks, collusion attacks, and physical-based attacks [17–19]. These incidents impact security properties including i.e., including confidentiality, integrity, availability. Also, these attacks can influence privacy attributes i.e. privacy identity, privacy access, and privacy disclosure [20]. From a security perspective, confidentiality is one of the security attributes associated with securing and privately controlled the users' data. This sensitive data can be only authorized to those who have the right to access, interact, and to be manipulated [19, 20]. Another security property is data integrity which responsible to afford the maintenance of user data during user transactions and communication, e.g., transfer, receive or store data from/to the cloud servers. Moreover, any violations related to the user data should be detected, e.g., data is missing, reformed, or cooperated during the intercommunication between mobile user services and real-time services in the cloud [16, 19, 20]. Since more user data will be hosted over the cloud and get a real-time exaction from mobile and other handheld devices, there are two challenges in data integrity have been mentioned by [21]:

Huge user data capacity makes predictable hashing structure not viable; thus, leads to losing data.

Integrity checking can only be applied when there are further requirements, for instance, no integrity guarantees and distributed setting for data unless there are dynamic operations.

Additional to data integrity, availability is a crucial security factor that supports access to different services, data, and real-time tools. The service provider should make sure that the service's tools used by users are a variable. However, when a certain service cannot be accessible, or the quality of service cannot meet the user's specifications, that may fix out the issues and maintain possible solutions or replace them with perfect [16, 19–21].

Object reusability and data remanence are a subcategory of data availability. From the privacy perspective, privacy is a critical concern while using mobile devices, user sensitive data exist and shared among different distrusted data servers, which are hosted and maintained by the service providers. Therefore, there are possible hazards that the confidential data, e.g., financial data, user photos, emails, and other secret contents are released to the public, business competitors, or even have insider user threats, external attacker threats, and data leakage. One of the offensives that target mobile devices is the WannaCrypt virus, which has attacked many devices around the world. Where more than 200,000 systems have been contaminated in over 150 countries. The virus force victims to pay a ransom to regain access to their file or systems [22, 23].

This research reviews the studies that have been analyzed in different perspectives based on Android security fields, investigate the Android permissions system that mobile users grant to install Apps, and during the usage of the Apps. It is worth to mention that in this article, security in mobile platforms has been discussed in which users can check out what are those apps could teach their data and take actions based on logs.

The rest of the article is organized as follows; Section 2 presents an overview of the Android system and its security mechanism. Section 3 provides the research methodology and Section 4 provided a general overview of the Android platform and application availability. Section 5 Android Date and Information Flow Control. Section 6 presents Android's existing security, privacy approach, and related works that aim to help users to protect their data using different data and information flow analysis methods. Section 7 discusses existing approaches in respect of analysis mechanisms. Finally, Section 8 provides the research discussion, recommendation, and conclusion.

2 Android Software Stack and Security Mechanism

Android Open-Source Project has provided Android as a modern operating system for mobile phones and tablets supported by the Open Handset Alliance (OHA). Due to Android process management, efficient memory, robust driver model, and core services networking support, it has been developed on top of the Linux Kernel. The Linux Kernel has been modified specially for the embedded environment. Android has two runtime environments Android Runtime (ART) and Dalvik Virtual Machine (DVM). Android Runtime environment uses ahead of time (AOT) compilation, e.g., apps compiled at the installation time to ready to state. This improves the overall

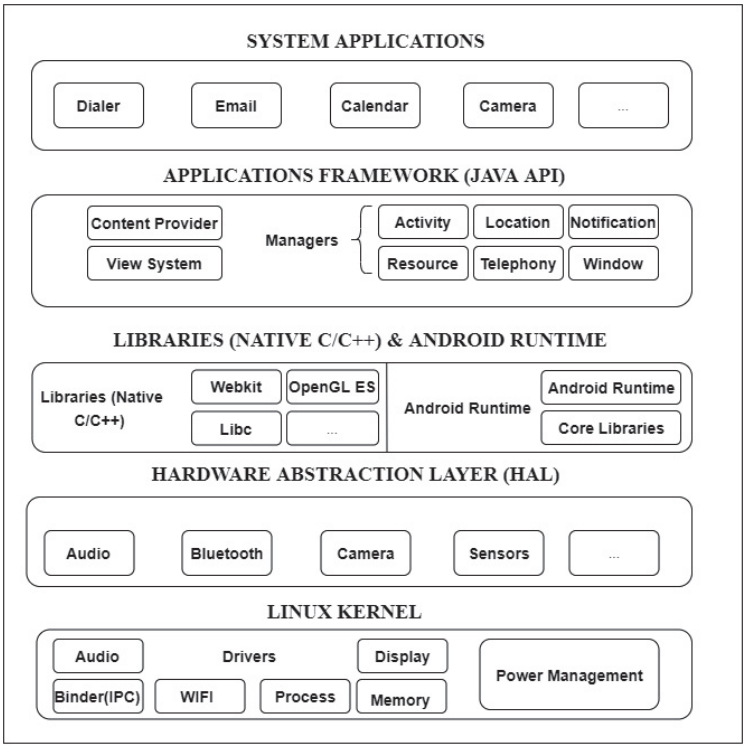

Figure 1 Android software stack layers.

performance and device battery life. However, Dalvik Virtual Machine uses just on time compiler as Android apps dex file that translated to their native representation on demand. Java programming language is used for Android apps development, and its shared libraries and native code are developed in C/C++ [19, 24]. Figure 1 illustrates the Android Software Stack Layers. At the core of the Android architecture, the security model provided by Google to protect apps and user data is a permission-based structure. This mechanism by default rejects any external access to features or functionality that could negatively impact the user experience, the overall system, or other applications installed on the device. However, google force android developers to declare these permissions during their development phase. These permissions allow users and apps to interact with different features, e.g., access to internet connections and GPS functions, personal information, system hardware and settings, and many other device features. Moreover, permissions are imposed by android at runtime but must be authorized by

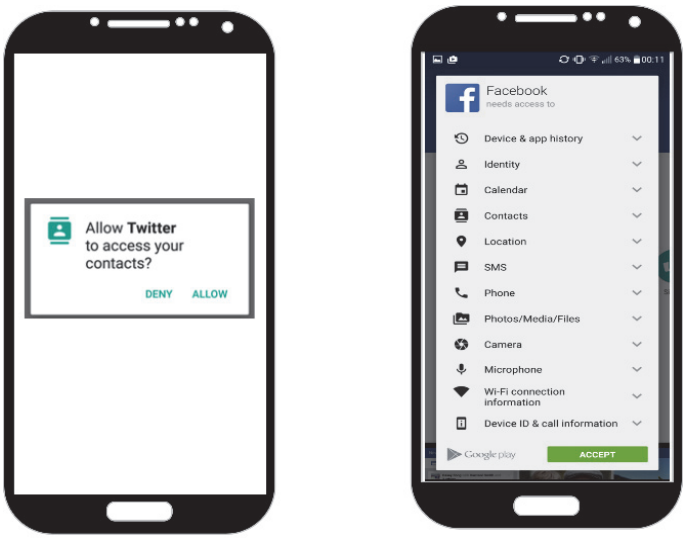

Figure 2 (Twitter and Facebook) used the concept of Android permission system at runtime.

the user at installation time. When users install a new application, the system prompted a set of permissions to be accepted or denied [25, 26]. The Android permission mechanism is fundamentally a mandatory access control system based on permission labels, which verifies whether an app has the required permissions when attempting to access protected resources [26]. Thus, the permissions approved for each application are well-defined in its required manifest file. The manifest file holds all significant values that are bound to the application at compile time cannot be altered afterward unless the application is recompiled, e.g., the name of the android app package, version of Android API, minimum and maximum Software Development Kit (SDK) that the app runs on. In Android 6.0 upward, users are prompted about the apps permissions at runtime, where the system allows users to grant and block the apps permissions groups individually and not as a set [27]. Figure 2 demonstrates two various mobile applications (Twitter and Facebook) that used the concept of the Android permission system at runtime.

Android permissions are classified into three security levels based on the sensitivities: Normal, Dangerous, and Signature. Normal permissions secure the API calls that do not damage the user data, e.g., set wallpaper; these do not involve user approval. This type of permissions is granted it by default in all apps without the user interactions [28]. On the other hand, dangerous permissions let an application perform harmful actions, e.g.,

record audio/video and get the storage contacts. Also, dangerous permissions are grouped according to the related functionality, i.e., the SEND_SMS, WRITE_SMS, RECEIVE_SMS, READ_SMS, and BROADCAST_SMS permissions comprise the "SMS" group. Moreover, signature permissions control access to very dangerous privileges, e.g., clear user data [26, 29–31]. Furthermore, Google provides another security model which can be realized by imposing each application to perform its functions within its secure sandbox. This technique designed Android as apart from other operating systems present in the market. Therefore, an instance of an application is isolated from other applications in memory [19, 32]. Android sandbox and Android permission system consider as the prominent security mechanisms. Besides, other security methods are used to increase the security and privacy of Android apps such as application signing which allows identifying the developer of an app. Application signing is a unique step used to ensure that the developed app is signed with certificates. This certificate is associated with a unique UID in the Android application sandbox. Each app installed on Android OS runs in a different UID and the system verifies that the app has been properly certified. Figure 3 shows a sample of the Android app Keystore that was used to create the app signing certificate.

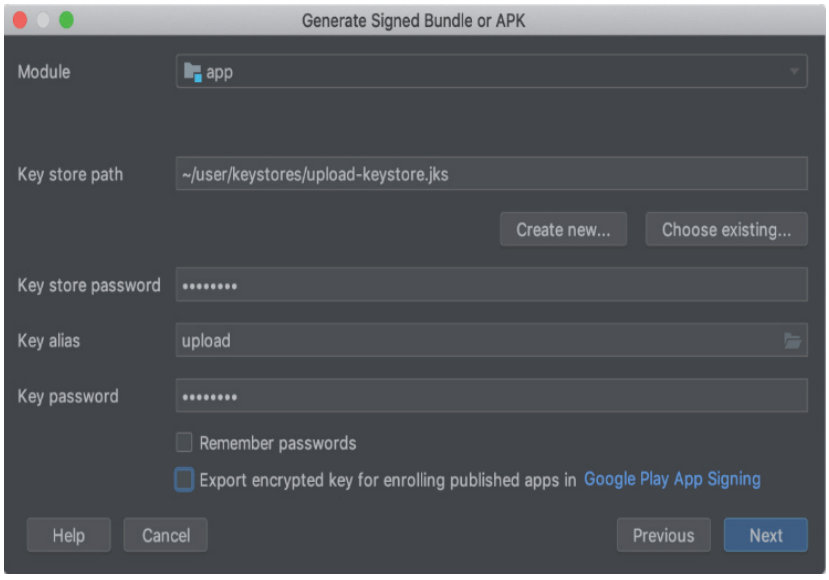

Figure 3 Android app keystore that used to create the app signing certificate.

Moreover, the Android platform provides secure inter-process communication where processes can communicate with each other through the binder, intents, services, and content providers [6, 28, 33, 34].

3 Methodology

This research is a systematic review study (SRS) considering all research done on the area related to privacy in mobile application environments. The research focuses on mobile applications and particularly on user privacy and data flow control for Android applications. This systematic review study is based on the Kitchenham's guidelines [35] that has already used by other systematic review studies [36–38]. In order to achieve the study objectives, the methodology is divided into different steps:

- Defining the research questions.
- Then the search keywords specification.
- Search process.
- Elimination criteria to consider only the relevant articles.
- Approach level classification.

Figure 4 illustrates the Systematic Review Study (SRS) Process.

3.1 Research Questions

The first step is about defining the research question. This systematic review study aims to address the following research questions:

- What are the state-of-art works done on user privacy and data flow control in Android apps? The answer to the research question, need to survey various issues and all approaches related to user privacy and data flow control in Android apps.

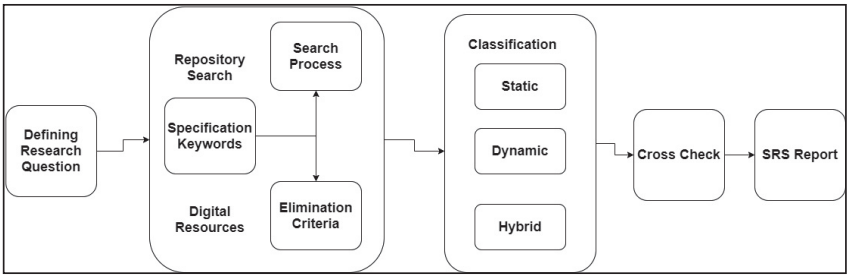

Figure 4 Systematic review study (SRS) process.

Table 1 Search keywords

No	Keywords
1	User Privacy; User-Privacy;
2	Data Flow; Dataflow; Data Flow Control; Data-Flow-Control;
3	Android; Android apps; Android-apps ;
4	Mobile; Mobile apps; Mobile apps ;
5	Smartphones; Smart Phone; Smart-Phone;

- What are the fundamental techniques and approaches used to control user privacy in Android apps? In this question, the research is focused on Android-specific characteristics that should be considered to achieve privacy and data flow control in different approaches level including OS, application, and user level.
- What challenges and open questions remain need to be addressed? With this question, the researchers investigate deeply the issues and challenge those related to data flow control and user privacy in Android apps. The researchers investigate the remaining open research questions that do not benefit from different research efforts.

3.2 Search Keywords Specification

Based on the research questions specified in Section 3.1 the search keywords were summarized. Table 1. Represents the actual selected and used keywords. The keywords 'Android apps', 'user privacy', and 'data flow control 'were used as the main broad search strings. The search string S is formed as a disjunction of the first two lines and conjunction of the disjunction of the last three lines of the specified keywords:

$$S =: L1 \text{ OR } L2 \text{ AND } (L3 \text{ OR } L5 \text{ OR } L5)$$

Whereas each line represents a disjunction of its selected keywords e.g., L1 =: {User Privacy OR User-Privacy}

3.3 Search Process

An electronic database literature search has been conducted using different digital resources such as Google search engine, IEEE Xplore, Web of Science, CiteSeer, ACM Digital Library, and SpringerLink.

3.4 Elimination Criteria

Research articles were selected based on their relevance as indicated not, only by the title but also by the abstract and in some cases scan and analysis of the full paper. Additionally, a manual examination of article references was conducted to select additional articles that might be missed using different digital resources. Peer-reviewed journal articles and peer-reviewed conference papers were included. However, only the subject books, book chapters, journal articles, and conference papers in which user privacy and data flow control for Android apps were explicitly discussed and described were considered eligible. Other resources that dealt with data flow control and privacy issues in mobile applications, in general, were not considered.

3.5 Approach Level Classification

This systematic review resulted in 109 different types of articles. These articles were analyzed and classified according to their approach level: operating system, application, or user level. Only articles that considered user privacy and data flow control explicitly with the respect of different approach levels were included. Hence, general articles that provide and discuss general ideas, research proposals, ethical and legal discussions were not considered. The research finally, resulted in 45 Android security approaches that accounted for 9 approaches that used a hybrid analysis mechanism to trace the privacy violation, 20 approaches that used a Dynamic mechanism, and 16 approaches that use a static analysis mechanism. These approaches have been built to trace the different privacy violations in Android OS and Android applications.

4 Android Platform and Applications Availability

Modern mobile technologies improve access to private and sensitive data. Currently, mobile devices present detailed personal contact information, a list of contacts, e-mail messages, appointments, location data, and much more personally identifiable information. Android is the superior operating system for mobile devices; it currently has the largest installed due to:

- It supports an enormous number of different devices including watches, tablets, TV sets, etc.
- It offers end-users a large variety of applications for completing their daily requirements through its official market.

- It supports mobile apps, developers, with different tools to develop an android application and publish them in the google play store.

In May 2015, the Apple app store had around 1,400,000 Apps, Amazon Appstore has 360,000 Apps, the Windows Phone store has 340,000, the BlackBerry world has about 140,000, and the Android google play has around 1,600,000 (Jason Hong, n.d.). The number of available applications in the Android Market (Google store) surpassed one million applications in July 2013 and was most recently reached two million applications last February 2016. In March 2017, the applications uploaded to Android google play were 2,800,000, comparing to the apple store which consists of around 2,200,000. In 2020, Google Play had more than 2.96 million available apps in the first quarter of 2020 [39]. Usually, user-installed apps have access to sensitive data and consider privacy less serious than users' expectations. Android Nougat (Android 7.0) split down on permission sharing between apps to save users' data. Moreover, android Nougat can no longer prevent users from uninstalling apps since Android 4.4 KitKat google enables application developers to ask users for permissions on a case-by-case basis to install these apps. This means that an application can grant access to storage and deny access to the camera. With these new updates, after applications are launched, the user is provided with a pop-up message as access permission to a feature. Hence, there is no limitations are placed to control the propagation of the user information, e.g., App X may have access to the user's location but no access to the Internet. App Y may have access to the Internet but no access to the user's location. This inoffensive situation may become threatening when App X hands over the location to App Y which may propagate it to a server on the Internet. Therefore, this research focuses on surveying how users can control and observe the app's traffics utilizing which app is using the users' services or data and log to check the types of inter-communication. Furthermore, investigate the designed monitoring mechanism and user interfaces that may enable users to understand what the application will do during loading and runtime.

5 Android Mobile Application Date and Information Flow Control

Different approaches, models, frameworks, and tools have been proposed for the Android permission system. These solutions followed various information flow approaches including Static Analysis and Dynamic Analysis.

Besides, some approaches improve the results using dynamic analysis with a combination of Execution Monitor, Program-Rewrite, and Program Slicing techniques. Thus, will assist the researchers to come up with more accurate evaluation results. Before highlighting different flow control approaches, there is a need to consider the differences between Date Flow Control (DFC) and Information Flow Control (IFC). DFC is simply a term used to describe whether an application under IFC protection has misled outside the expected flow. When the program runs, it is instrumented in some mode to ensure that each instruction or set of instructions does not violate data flow integrity, i.e. the values which were expected to be untainted by the operation have remained so [40].

On the other hand, the flow of data between different entities can be defined as a process of communication between two nodes where data transmission should be secure. The DFC is a mechanism for the receiver to monitor the transmission speed so that the receiving node is not overwhelmed with data from the transmitting node. Flow control should be recognized from congestion control, which is used for managing the flow of data when congestion has occurred. There is security leakage produced by illegal data flow between entities. For instance, in a healthcare institution, a patient's private information can be leaked by the hospital servers and transmitting via network channels to other medical providers, whereas there is a security leak caused by the illegal operation on the received data by the entity. The privacy exposure is caused by illegal operations of the healthcare institution server performing numerous statistical analyses or modifications of the received patient's information [41].

IFC focuses on the flow of confidential user information. User information is a transfer, broadcast, and process during the execution of untrusted programs or software systems to make sure that the method of transmission will be operated securely [42–46]. Information flow is not restricted to a single component of an app but occurs frequently between components of the same context and even various apps [47]. There are two main related aspects of having secure information flows are (a) information confidentiality and (b) information integrity. This means, no private information should leak e.g., over network and cloud services and no untrusted involvement from the network should leak into a database or related storage. There are various IFC approaches have been developed [47–50] to protect user information.

The DFC and IFC have been analyzed in Android mobile apps using different mechanisms, including static and dynamic analysis. The next section focuses on data and information flow analysis techniques.

6 Android Mobile Application Data and Information Flow Analysis

Researchers and developers used some analysis methods to trace and monitor the flow of data and information, detect vulnerabilities and privacy violations, errors, and bugs before the app become available to users.

The static analysis mechanisms are used to analyze the applications in an isolated way, use the data and information flow examination of symbolic execution to determine a priori whether an app will leak privacy information or not without running the program. Static analysis exam all possible execution tracks including those paths which already been executed and detect the malicious behavior in the code segments [6, 18, 26, 38, 44, 51–54]. However, [15, 55] discussed that a static analysis method is a complicated approach for android mobile applications due to three main reasons: (1) Android applications consist of different components, i.e. activities, services, broadcast receivers, and content providers. The communications between these components involve Intents and Intent Filter which cause a discontinuity in the control-flow of Android applications and lead to pre-processing of the code to resolve links between components. (2) The issue related to user behaviors while using the touch screen action and interacting with the GUI. The management of user involvement can be controlled via handling specific call-back methods/functions, e.g., an on Click method which is called when the user clicks on a button. Hence, static analysis requires an accurate model that can stimulate users' behavior very obviously. (3) The lifecycle of Android components, due to there is no main method in Android programming. Android system uses different components states lifecycle by calling call-back methods such as onStart or onResume. Nevertheless, these lifecycle procedures are not directly connected to the code. Thus, modeling the Android application permits to connect call-back methods to the rest of the code [15, 55].

The second type of analysis is dynamic information flow analysis which is concerned with monitoring, tracking, and regulating a program execution during runtime. This type is more precise than static analysis because it requires the current execution of the program to reach appropriate code coverage; it does not leak information and can also cover language features, e.g., pointers, arrays, and exceptions easier than static analysis [44, 56–58]. Also, the researchers may perform a combination of other analysis techniques such as, program re-write techniques that are considered as one of the runtime approaches. This approach focuses on satisfying security policy during

runtime, its expensive process, and suffering a non-negligible overhead on the applications. The core feature of this method is rewriting instructions that interrupt by security policy at runtime. Comparing with other approaches, program-rewrite fail to detect implicit flows of information, which can be achieved by a static analyzer that considers the whole code, rather than just the executed instructions [59]. Another important runtime approach is the execution monitor approach. This technique achieved data labeling and information tracking during the execution of the program. It detects, monitors and follows different events when applications deal with the data coming/going from/too sensitive information sources, e.g., device location sensor, user phone contacts, then reporting these applications, data type, and network destination to the user [45]. [44] discussed another mechanism called program slicing. The technique gets involved especially in re-engineering or debugging the program, and it focuses on specific parts of a given program. There are different types of program slicing: (1) Static slicing which enhanced the static analysis by computed symbolic values without considering the program input. (2) Dynamic slicing, in the technique the slice is calculated for fixed input or data value. (3) Forward slicing is focused on program statements and what are those variables value affected the statements. (4) Backward slicing, this type of slicing is calculated from any point in the program to discover all statements that can affect specific variables [44]. Therefore, the researchers have reviewed different tools that used the approaches mentioned above.

Different approaches are proposed to support the user's privacy and avoid data violations. Most of these approaches used the concept of static and dynamic analysis. Table 1 summarized the differences between the two types. The next section discusses related studies and approaches that support user privacy protection tools using static and dynamic analysis techniques.

7 Related Work

Android apps use permission systems through pre-defined permission metadata in the Android manifest file. Android components can be protected with a permission system or custom permission to secure the access of apps to sensitive data and private resources. Developers are required to set the permission attribute to access apps services APIs and methods i.e., storage methods, network APIs, camera access. However, some apps can evade the permission system and gain access to protected resources without user consent. Thus, can occur through covert and side channels that used the permission system limitation to attack user's privacy. Inside channels, apps

Table 2 Comparison between static and dynamic analysis

Components	Statics Analysis	Dynamic Analysis
Target code execution	Not possible	Possible
Type of execution	Code investigation – No runtime operations	Runtime investigation and operations
Time required	More time required to go through the lines of codes	Less time since using an automation method
Input Type	Binary files, scripting language files, etc	Memory snapshots, runtime API data
Advantage	Fewer resources and time Exam all possible execution tracks through one file: Android manifest.xml. Can help to detect many vulnerabilities i.e. private data leaks, unauthorized access to protected or private resources, and intent injection	Deep examination and higher discovery rate with obscure malware spot More precise due to the current execution of the program to reach appropriate code coverage
Disadvantage	Constrained signature database and can identify only within the scope of the known malware types	Power consumption and more time to perform the process
Accuracy of the results	Low rate of accuracy compared with dynamic analysis	Improved than static analysis

gain access to protected resources and bypass the permission system; whereas covert channels support the interaction between two communicated apps so one app can allocate its permission protected data to be used by another app that lacking those permissions [60, 61]. Furthermore, many studies discussed the possible solutions and approaches to trace and avoid privacy violations. The below sections highlighted the approaches introduced using three main analysis techniques:

7.1 Android Existing Privacy Approaches Using Static Analysis

Static analysis approaches have been recommended for various evaluation methods, such as assessing the security and privacy violation of Android apps, tracing app clones, and presenting test case generation. Most studies have been proposed to attempt a solution for one or more of the numerous challenges that program analyzers cross when dealing with Android apps. In

2020, [62] proposed a static data flow approach called DroidRista that is used to detect the flow of sensitive data in the Android mobile app. DroidRista evaluates the ICC, reflection, and implicit flow in Android apps [62]. Furthermore, MR-Droid developed for [63] purposes to detect inter-app communication threats and solved the problems of accurate and scalable of ICC. Nevertheless, the approach suffers from some limitations, i.e., the tool can operate intent-based ICC communications only where the security risks can be handled by other inter-app channels like content providers, shared preferences location-based, etc [63]. Moreover, [53] proposed a detecting Permission Over-claim approach which is based on byte code static analysis and semantic similarity analysis. This approach provides a detection method for all explicit, implicit, and Ad library permission overclaim problems [53]. In 2016, [64] presented a methodology to obtain privacy profiles for permission states. The Personalized Privacy Assistant (PPA) can manage the user profile in a personalized privacy assistant and allow users to configure their apps' permission settings. However, the PPA is limited to generate a user-specific profile thus, does not reflect each application's access to private data on a case-by-case basis [65]. In the same year, [9] introduced an Android security solution targeting the inconsistent security enforcement within the Android framework. This approach provided a methodology that discovers the inconsistency in security policy enforcement in Android using a proposed static analysis tool called Kratos. However, Kratos is only a static analysis tool for systematically discovering the inconsistencies of security checking. The methodology did not discuss the protected resources in Android's application framework [14, 15]. In 2015, [66] implemented a static analyzer called EDGEMINER that explores the entire Android framework statically to produce API summaries automatically. These APIs address the issue of implicit control flow transitions including the well-defined call-backs in the Android framework. Also, the tool performs inter-procedural backward data flow investigation to extract a list of Androids OS registration-call-back. However, most of these mechanisms are based on the analysis of permissions granted to apps and only discussed the tracing of information flow based on the Android OS level. This type of analysis is not sufficient for detecting all levels of malware, because it can detect collision attacks on the OS level without considering the user control, as well as it may consider only the security methods that followed by the OS, i.e., Sandbox and Android configure permissions [66]. Besides, [67] introduced an IccTA which performs static analysis to detect privacy leaks between different components in Android applications. IccTA supports inter-component detection and improves the

precision of the analysis [67]. In 2014, [68] presented a FlowDroid tool that performs a static taint analysis for Android applications. FlowDroid analyses the apps' bytecode and configuration files to check out the potential privacy leaks caused by carelessness or produced by malicious [68]. In the same year, [69] proposed an Amandroid that focuses on providing a precise and general Inter-component Data Flow analysis approach for the security selection of android apps. This approach is limited to handle exceptions including the handle concurrency and reflections issue. In the same context, [70] implemented an android app called ProfileDroid. This tool takes care of multi-layer monitoring and profiling apps. This approach profiles the apps at four main layers: (a) static, or app specification, (b) user interaction, (c) operating system, and (d) network. However, its main obstacle is that it does not offer any profiling results about consumed time [70]. Besides, PermissionFlow proposed by [71] aims to provide automatic detection of inter-application permission infiltration in android applications. This static analysis tool is used to detect unauthorized access to private information by capturing the flow of permissions. In contrast, PermissionFlow does not detect the native code permissions. Also, the tool suffers from redundant checking of permissions and data-dependent which causes erroneous during the detection process. Finally, this tool uses implicit intents only which leads to negatives results for the apps that use the communication between other Android components [71]. In 2012, [72] developed a static analyser tool called SCANDAL. Their developed tool detects privacy leaks in Android applications and determines the flow of data from the source of information until reaching its target channel. SCANDAL covers limited privacy information includes location, phone, SMS, and Eavesdropping [72]. In the same year, [73] introduced Constroid that provides a data-centric security policy management framework for Android. Constroid provides partial enforcement of fine-grained and allows users to control the access to policies in Android. This approach modified the content provider and Inter-component Communication Channel (ICC). However, Constroid defined the security policies for each Android components, instead of specifying permissions for each app. Furthermore, these policies are specified by the user, not by the developer. Though, this approach can easily confuse users as they are held responsible for requiring security and privacy policies [73]. Furthermore, [74] developed TrustDroidTM that supports the static analysis concept to track the app communication and prevent leakage of sensitive information in a user's mobile phone. However, this approach has some limitations including, the tacking of tainted data that should be written to a file otherwise, the process

of monitoring will not be done. Also, the tool starts detecting and tacking the app while they are loading means no monitoring for apps during the installation and even during the usage [74]. In 2011, [75] developed a ComDroid approach that observes the application communication vulnerabilities. ComDroid can be used by developers to investigate their applications before its final release. Nevertheless, ComDroid detected android Intent control flow across functions and did not distinguish between paths that used control flow statements. Also, it does not track the privilege across pending Intents and Intents that performs URI read/write permissions [75]. Additionally, [76] proposed AppInspector which is an automated security validation tool. It performs a static analysis for apps and generates reports of potential security and privacy violations. Besides, they proposed a dynamic method that tracks an app's use of sensitive information and checks for suspicious behavior such as excessive resource consumption or deleting user data [76]. However, this approach has less scalability of the app base which leads to an increase in the malicious apps in the market.

The tool performs a single party to achieve the entire process in the android app, e.g., tracing high privacy information flow, detecting security and privacy hazards, and reporting potential risks of information misuse [77].

7.2 Android Existing Privacy Approaches Using Dynamic Analysis

Due to the limitation of static analysis, some approaches have been proposed with the usage of dynamic analysis. This analysis traces the behavior of the app by running a specialized benchmark while the code is running. In 2020, [78] proposed a CenterYou framework that utilizes a pseudo data technique and a cloud-based decision-making approach to examine and protect Smartphone devices from over-requested permissions by installed applications and identifies potential privacy leakages. However, CenterYou requires root access and Zygote to enable the injection of the services [78]. In 2019, [79] proposed Android Flexible Permissions (AFP which provides a user flexible permissions management approach). This approach aimed at supporting end-users to control and manage Android permissions. Users who installed the apps with AFP configuration can identify and customize fine-grained permission levels on private or confidential resources [79, 80]. In the same year, [81] introduced a dynamic analysis approach that monitors the permissions requested by apps during the run-time and recognizes those requested permissions by the app's fundamental functionality from those

demanded by third-party libraries linked with the app [81]. In 2016, [8] proposed a framework that enforces fine-grained security privacy policies and enables users to manage access of applications to sensitive elements. This approach allows users to modify their security restrictions dynamically at runtime without the need to recompile or to reinstall the apps. The framework does not have a clear background of the users regarding the information-flow path which may lead to privacy violation [8, 82]. In the same year, [82] introduced a dynamic analysis tool that generates inputs for the Dynamic Analysis of Android Malware. IntelliDroid can produce a small set of inputs that allow the dynamic analysis to decide when it is malicious in android applications. IntelliDroid does not operate with implicit Intents, Content Providers, and native code execution. Also, IntelliDroid partially handles reflection as it cannot recognize the path constraints after the reflected call [24, 82]. Moreover, [83] proposed DroidDisintegrator which tracking the inter-component information flow and tracing of component resource use in apps. Also, it validates the feasibility of the component-level of information flow control while controlling the app behavior [13, 84–86]. However, the tool uses only dynamic analysis while the tracing of the internal logic of sensitive behaviors should take place. Hence, the static analysis is needed as well to perform better tracking for information, decrease falsely reported flows, and detect decomposable flows. Moreover, [56] described a cross-platform technique called ReCon which exposes personally identifiable information (PII) leaks over the network. The system extracts user location as a sample and enables users to control their information about privacy leaks from the network, provide feedback about appropriate leaks, and they can as well change the information sent to third parties [56]. In 2015, [87] proposed a dynamic tool called Inspeckage that builds under the Xposed Framework. Inspeckage provided investigating for mobile apps' network traffic and information gathering includes app permissions, third party libraries, debuggable status of the apps, and export components [87]. In the same year, [88] implemented COMBdroid (Covert Malware Blocker) approach that discourses Android security concerns by enforcing fine-grained and allow users to define policies. COMBdroid adjusts an application before installation instead of modifying Android OS and permitting it to override security vulnerabilities at runtime. On the other hand, the approach capability is restricted with only three policies. Many threats should be considered in the overall security checking [88]. In 2014, introduced the TaintDroid approach that was used to monitor the flow of user's private data that have been developed/downloaded from third-party stores [13]. However, this tool traces only explicit data flow, in which

a bytecode directly transmits information from its source objects to its destination objects. Moreover, it cannot detect malware in apps that have their libraries [88]. Also, [89] introduced the VetDroid approach for remolding sensitive behaviors in Android apps from a novel permission use perspective. This systematic approach is used to efficiently structure permission use behaviors, e.g., how applications use permissions to access privacy system resources, and how the application further utilizes these attained permission-sensitive resources. VetDroid can be suggested to be used in finding more information leaks than TaintDroid proposed by [89]. In 2013, [90] introduced an AppsPlayground approach that supports the dynamic analysis of Android mobile applications and track privacy leakage. The tool required a modification over the Android software stack [90]. In the same year, [91] presented a new analysis AppIntent framework that provides a sequence of graphical user interface operations to lead the data transmission. Thus, helping the specialists to control if the data communication is used intended or not. AppIntent has two main limitations; firstly, the tool does not support native codes. As well as, since the Android InstrumentationTestRunner method does not support the instrumentation of network input, the tool cannot simulate network inputs generated by symbolic execution [91]. Moreover, [44] introduced a new approach which provided a dynamic and operational information security solution. This approach supports the interaction between the user and the security process where users can manage the security of their applications without defining intricate security policies before running the mobile application. However, this approach does not enable the application user to decide which policies should take place when the information starts sharing with the cloud. Moreover, there is no grant to users about the flow of this information and personal data [44]. In 2012, [11] presented a systematic approach for the detection of malicious applications named DroidRanger. This approach implemented two main schemes: a permission-based behavioral footprinting scheme and a heuristics-based filtering scheme. The tool focused on system calls used by the existing Android kernel or made with the OS privilege. Therefore, these two heuristics are not sufficient to detect all malware in android markets, and it's only limited to detect variants of common malware types or uncommon malware that dynamically load untrusted code. Also, DroidRanger performs an offline investigation to detect malware in Android Markets, thus no runtime detection of apps malware [92]. In 2011, [86] established a security approach called AppFence. This approach modified the Android operating system to execute privacy controls on existing Android applications. AppFence allows users to keep tracking the private information

and data that transfer from their device to a third-party application. However, AppFence has numerous limitations: (1) AppFence suffers significant performance overhead; thus, due to the taint tracking on every single Dalvik bytecode execution. (2) AppFence required firmware modification, therefore, deploying the tool on multiple Android devices can be challenging [93]. (3) AppFence has some blocking feature, e.g., does not detect information leaked through control flow operations [86]. In 2010, [94] proposed an Apex framework that introduced a policy enforcement mechanism for Android. It allows the user to selectively grant permissions to applications as well as enforce constraints on the usage of resources. However, Apex is limited to some of the available permissions [95]. [96] proposed Android modified version called MockDroid. This framework is used by users to 'mock' applications' resources access. MockDroid offers an extended permission technique where the resource access is not blocked but results in empty, unavailable, or dummy data [96] MockDroid'. In 2009, [97] presented the CRePE approach that permits context-related strategies to be well-defined either by the end-users or apps market owners. Context-related policies are security policies that require the responsiveness of the context of the phone which can be defined using different statuses, e.g., location, time, temperature, noise, and light. On the other hand, CRePE only checks the context of active policies. Nevertheless, other permissions and policies should be considered as well [97]. In the same year, [98] proposed SCANDROID that achieve incremental checking of android applications extracted security specifications from applications which manifest and checks whether data flows through those applications are consistent with its permissions [99]. This approach is limited because it cannot analyse the full packaged Android applications and it does not yet apply to real-world applications [98].

7.3 Android Existing Privacy Approaches Using Hybrid Analysis

Because of the limitations of static and dynamic analysis methods, many approaches prefer to perform both static and dynamic analysis to obtained accurate results. In 2020, [100] proposed an approach called Alde that identifies the analytics libraries integrated with Android apps. This approach uses both static and dynamic analysis to detect the users' in-app actions gathered by analytics libraries. However, Alde is not handling the process of Android inter-component communication and inter-process communication, which might failure some effects. Also, the approach does not perform the analysis on a large-scale [100]. In 2019, [54] introduced an automated

vulnerability detection model that builds both static and dynamic analysis methods [54]. AndroShield Cannot run different versions of an app under different profiles [54]. In 2018, Ostorlab [101] is a cloud-based approach that provides static and dynamic analysis to identify the apps privacy leakage and vulnerability in mobile apps. The free version of Ostorlab does not offer any dynamic analysis features. Although it lacks providing the source code, modules, or architecture [101]. In 2016, [102] proposed a security analysis model that can make up for their shortcomings with each other, enable the analysis of malicious behavior more comprehensively and accurately [102]. In the same year, [103] discussed the malware characterization that was implemented in the Android manifest file. Moreover, researchers gave the user the ability to improve the efficiency of Android permission which can inform the user about the risk of Android permission and apps [103]. In 2014, [104] proposed an AppFork which allows users to isolate and secures two different entities on a single phone single, i.e, work, and active personal profiles. The researchers address data leakage channels by developing a ChannelCheck tool that uses to detect the leakage of channels automatically. AppFork still realizes the security of virtualization-based methods, but with a smaller overhead [104, 105]. In the same year, [106] developed a Cassandra app checker tool. The developed tool permits users of mobile devices to check whether Android apps observe their privacy requirements before installing these apps. Moreover, Cassandra allows the user to define security policies, verify that apps follow these policies before installation, and performs the security analysis of apps on a server. However, Cassandra covers around 211 out of 218 Dalvik instructions. Also, it does not support exception handling and synchronization methods [106]. In 2012, [107] discussed the application development of the Android mobile platform and provided an important layered approach that could be used to secure the information in Android OS. Their layered method guides them to discuss the details of an android app called "AASandbox." This app was developed by [12] and can achieve both static and dynamic investigations on android programs to reveal suspicious [107] automatically. On the other hand, AASandbox generates low detection accuracy because it is very varied [65, 108]. In 2006, [109] provided an information tracking mechanism based on resource classification. They modified the Android kernel of integrated wireless devices so that processes and files are labelled to regulate access to different system resources [109].

Most of the previous studies do not put the control of the apps' behaviors in the user's hand, however, all these proposed approaches and tools are

installed in the user device to detect or monitor the flow of the other app data or detect the permissions used in these apps.

8 Discussion

Existing solutions have used different control flow approaches to address privacy in mobile systems. However, they are limited because they do not have perceptibility into network flows from mobile devices, the ability to adjust the application permissions, and the deployment model that facilitates large-scale adoption to ensure broad impact. At first, glance, addressing all these limitations seems to impose a high barrier to success, due to the difficulty to address security issues that may occur during the interaction between the user and different services provides. Moreover, an evaluation approach that combines static and dynamic analysis to assess any given application may have low speed especially with static analysis as well as other different mentioned issues. The user can be offered, the control of the application permissions, where they can change and modify the permission rules. Tables 3–5 illustrate Android approaches that used different analysis concepts including static, dynamic, and hybrid analysis.

Moreover, a user prefers to have a level of security approaches that support user control over the privacy and the violations that may leakage the private data. As mentioned in Section 7, many of the proposed approaches use static or dynamic analysis to control the flow of information and user privacy. Most of the discussed methodologies do not support native codes and do not focus on the user's actions that need to be performed when flow policies are hacked. Furthermore, some of these approaches start their detecting process during app loading time, thus no app monitoring during installation time or usage. Users have no control over sensitive or private data, in such a way that the user can change or modify the way that data processed or manipulated. From the previous tools and frameworks, it can be observed that most of the approaches try to be convenient with securing mobile devices by producing and implementing tools that monitor installed apps behaviors in users' devices, without considering how a user can control these apps. Besides, these approaches do not invigilator the app after installation time and during the usage by users. Some of the proposed approaches detected the malware statically, thus, no real-time usage and it may affect the accuracy and accuracy of tracking the user's private data. Moreover, no such approach provides a clear setting for users to allow them to manage the device resources with certain permissions. Most of the permissions control is done in the app's

Table 3 Android proposed approaches that used static analysis concept

No	Year	Approach	Mechanism	Limitations
1	2020	DroidRista: is an Android approach that provides a static data flow analysis and detects sensitive data leakage.	Static	Do not analyze apps that support native code.
2	2017	MR-Droid: a scalable and prioritized static analysis tool of inter-app communication risks	Static	Cannot handle other inter-app channels like content providers, shared preferences. Its focus only on intent-based ICC communication.
3	2017	Detecting Permission Over-claim: a novel approach based on byte code static analysis	Static	high memory usages which may cause false results. The approach considers only the java codes.
4	2016	Privacy Profiles and Preference Modeling (PPA): methodology to allow users to configure the installed application permission.	Static	Limited to generate a user-specific profile. Do not reflect each application's access to private data on a case-by-case basis.
5	2016	Kratos: Determining Inconsistent Security Policy Enforcement in the Android Framework.	Static	Only a static analysis tool for systematic tracing inconsistencies in security enforcement. Did not make the effort to classify protected components in Android's application framework.
6	2015	EDGEMINER: explore statically the entire Android framework to automatically produce API summaries	Static	Detection of collision attacks in OS level without considering the user control
7	2015	IccTA: Detect privacy leaks between different android components.	Static	Cannot detect a leak through multi-threading. Can miss leaks through native calls that their rules model incorrectly.
8	2014	FlowDroid: Precise context, flow, field, object-sensitive, and lifecycle-aware static examination for Android apps.	Static	Misses different private and sensitive data leaks from benign off-the-shelf Android applications

(*Continued*)

Table 3 Continued

No	Year	Approach	Mechanism	Limitations
9	2014	Amandroid: A Precise and General Inter-component Data Flow Analysis Approach.	Static	Limited ability to handle exceptions including the handle concurrency and reflections issue.
10	2014	ProfileDroid: multi-layer monitoring and profiling app.	Static	High overhead. No details about operations consumed time
11	2013	PermissionFlow: Automatic detection of inter-application permission leaks in Android applications.	Static	Does not support the security detection for native code permissions. The use of implicit intents only leads to negatives results for the apps that use the communication between other android components.
12	2012	SCANDAL: detects privacy leaks in Android applications and determines the flow of data from the source of information till reaching its target channel.	Static	Does not support reflection-related APIs. Detect only three main APIs including Location Information, Phone Identifiers, and Eavesdropping (track both audio and video).
13	2012	Constroid: a data-centric security rule management schema for Android	Static	Confuse users as they are thought that the responsibility of specifying security policy and privacy rules depend on them.
14	2012	TrustDroid: tracking the app communication and prevent leakage of sensitive information.	Static	Tacking of tainted data that should be written to a file. Detecting and tacking the app while they are loading, means no monitoring for apps during the installation and even during the usage.

(*Continued*)

Table 3 Continued

No	Year	Approach	Mechanism	Limitations
15	2011	ComDroid: A detection tool used by developers to monitor their application communication vulnerabilities before the app release to the app market.	Static	ComDroid detected android Intent control flow across functions and did not distinguish between paths that used control flow statements. Does not track the privilege across pending Intents and Intents that performs URI read/write permissions.
16	2011	AppInspector: an automated security validation tool that used to examines apps and generates reports of potential security and confidentiality violations	Static	Less scalability of the app base. A single party to achieve the entire process in android.

development stage by android developers. Therefore, this research provides a recommendation to develop a technique that offers users of the apps the priority to change the apps' behaviors once it starts accessing the user data. The permissions provided by apps can be modified by the user, as well as the user can lock or even prevent them from accessing any private data. Also, researchers recommend that enables the user to control mobile application actions based on configurable privacy and data flow policy during runtime. The technique should consider any behaviors done by installed apps and notify the user about the tracks, thus help the user to detect which app access private data or any other information. To ensure that users know what they are doing? They should explicitly approve any data access or manipulation rather than leaving it to the platforms.

Assume the following scenario, that DS1, DS2, DS3 are data sources that will be processed using the mobile app. The monitoring mechanism that regulates and controls the flow of processed data during mobile app runtime has a user privacy policy stated that DS2 (Data source 2) must not flow to DT1 (Data Target 2) as shown in Figure 5.

Then possible data flow scenarios are:

- Data source (DS2) flowed to data target (DT2 or DT3) allowed based on policy rules.

Table 4 Android proposed approaches that used dynamic analysis concept

No	Year	Approach	Mechanism	Limitations
1	2020	CenterYou: is a framework that applies the pseudo data technique.	Dynamic	Requires root access and Zygote to enable the injection of the services.
2	2019	Android Flexible Permissions (AFP): a user-centric approach to flexible permissions management	Dynamic	Works with apps that configure the AFP setting
3	2019	REAPER: Dynamic Analysis for Augmenting the Android Permission System	Dynamic	Used to prevent a third-party privacy violation only.
3	2016	The framework that enforces fine-grained security privacy policies and enables users to control access of applications to sensitive.	Dynamic	Unclear to users the information-flow path that may lead to privacy violation.
4	2016	IntelliDroid: A Targeted Input tool for the Dynamic Investigation of Android Malware.	Dynamic	IntelliDroid does not operate with implicit Intent, Content Providers, and native code execution. The tool is not capable of generating inputs for encrypted and hashed functions.
5	2016	DroidDisintegrator: tracking of inter-component information flow and observing of resources use in apps	Dynamic	Performed only dynamic analysis while the tracing of the internal logic of sensitive behaviors should take place.
6	2016	ReCon: A cross-platform technique that exposes personally identifiable information (PII) leaks over the network.	Dynamic	Require network connection to detect and extract PII leaks.
7	2015	Inspeckage: is a dynamic Android application analysis tool that builds under Xposed Framework which makes dynamic analysis very easy.	Dynamic	Offers dynamic analysis only and requires root access using Xposed Framework

(*Continued*)

Table 4 Continued

No	Year	Approach	Mechanism	Limitations
8	2015	COMBdroid: Discourses Android security concerns by enforcing fine-grained and allow users to define policies, maintain a list of trusted and untrusted behaviors.	Dynamic	Users' preference list is recalled within the scope of one application.
9	2014	TaintDroid: Evaluate the flow of user's private data that goes through the third-party stores.	Dynamic	Traces only explicit data flow, in which a bytecode directly transmits information from its source objects to its destination objects.
10	2014	VetDroid: remolding sensitive behaviors in Android apps from a novel permission use perspective.	Dynamic	Needs a quite intrusive change in Dalvik VM, Binder, and Linux kernel in the Android system that restricts the simplicity to port to different Android system versions. It inherited the drawbacks of TaintDroid as it built on top of it.
11	2013	AppsPlayground: is an Android framework that automates the analysis of smartphone applications and tracks privacy leakage.	Dynamic	The tool modified the Android software stack to run the analysis.
12	2013	AppIntent: Analyzing critical data transmission in android for privacy leakage recognition.	Dynamic	Does not support native codes. Does not support the instrumentation of network input.
13	2013	A dynamic and operational information security solution.	Dynamic	Does not support the procedures of involving users in the monitoring process. No user involvement in deciding which policies that should take place when the information starts sharing with the cloud. No grant to users about the flow of this information and personal data.

(*Continued*)

Table 4 Continued

No	Year	Approach	Mechanism	Limitations
14	2012	DroidRanger: a systematic approach for the detection of malicious applications using a permission-based interactive foot printing scheme and a heuristics-based filtering structure.	Dynamic	Limited to detect variants of common malware types or uncommon malware that dynamically load untrusted code. DroidRanger performs offline analysis to detect malware in Android Markets.
15	2011	AppFence: keep tracking the private information and data that transfer from their device through a third-party application.	Dynamic	significant performance overhead. Required firmware modification. Does not track information leaked through control flow operations.
16	2010	Apex: extending framework for the android permission model and enforcement with user-defined runtime constraints.	Dynamic	Limited to some of the available permissions.
17	2010	MockDroid: extended permission technique where the resources access can be controlled by users.	Dynamic	Not available for end-user.
18	2009	CRePE: interrupt critical API calls and filter against predefined policies.	Dynamic	Checks only the context of the active policies.
19	2009	SCANDROID: extracts security specifications from application manifests.	Dynamic	Cannot analyze the full packaged Android applications

- Data source (DS2) flowed to the data target (DT1) was denied according to policy.
- Data source (DS1 and DS2) processed in parallel and DS2 copied to DS1 then DS1 flow to DT1 denied as stated in the policy.

Table 5 Android proposed approaches that used static analysis concept

No	Year	Approach	Mechanism	Limitations
1	2020	Alde: detect the users' in-app actions collected by analytics libraries	Static and Dynamic	Android inter-component communication and inter-process communication are not handled in the 'Alde' analysis process, which might failure some consequences. Also, the approach does not perform the analysis of large-scale.
2	2019	AndroShield: Automated Android Applications Vulnerability Detection[54].	Static and Dynamic	Cannot run different versions of an app under different profiles.
3	2017	Ostorlab: a cloud-based security and privacy scanner framework for Android or iOS[101].	Static and Dynamic	Do not provide a full dynamic analysis for free.
4	2016	Malicious behavior analysis for android applications	Static and Dynamic	Consider more sensitive APIs and provide the Android market real App for fans to use. An integration of other malware detection technique, e.g., dynamic taint analysis
5	2016	Investigation of Malicious Behaviour of Android Apps	Static and Dynamic	Monitoring Android sensitive API and explore the app's vulnerability.
6	2014	AppFork: Data leakage tool that isolates and secures partitions belonging to work and personal profiles.	Static and Dynamic	Cannot run different versions of an app under different profiles.
7	2014	Cassandra: Information-Flow Analysis	Static and Dynamic	Does not support exception handling and synchronization methods
8	2012	AASandbox: a layered approach to secure private information in Android OS.	Static and Dynamic	AASandbox generates low detection accuracy
9	2006	Using Labelling to Avoid Cross-Service Attacks Against Smart Phones	Tracking and Monitoring	Extend policy language to enable users to describe more labels for complex policies.

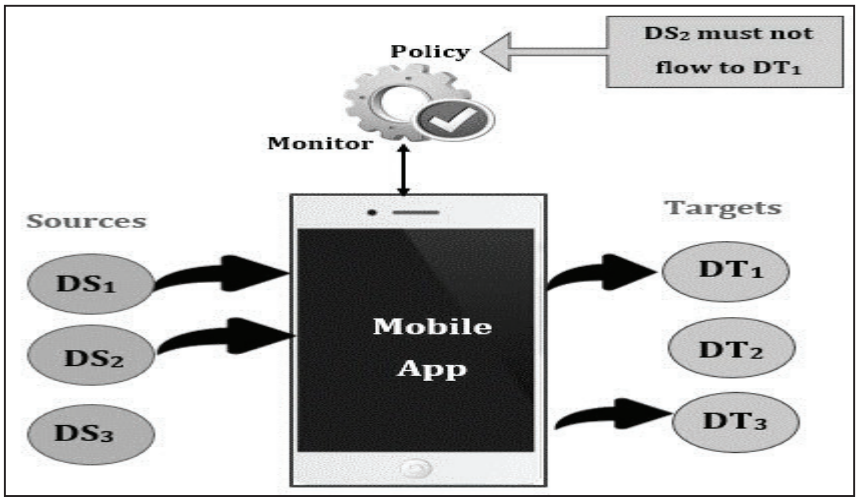

Figure 5 The flow of Apps while accessing the user information.

9 Conclusion

Mobile devices store personal information, e.g., user location accounts detail and sensitive data; thus, privacy and security of user device is a major concern. Android as an open-source OS secures apps by sandboxing app execution and enforces the apps developers to maintain a set of provided permissions. Moreover, different studies have been conducted to provide solutions that protect user's devices, most of these proposed approaches can monitor the behaviors of installed apps but not permit the user to control or prevent any insecure behavior. Thus, this article discussed exciting approaches designed to provide security approaches (tools and frameworks) that track the actions of installed apps. To summarize the state-of-the-art of user privacy and data flow control in Android apps, this article systematically reviewed published approaches. In this systematic review process, 109 articles were collected. The article discusses general concepts, proposals, ethical and legal discussions that were not considered. We have found that no such mechanism allows users to control mobile application behavior based on configurable privacy and data flow policy during runtime. Finally, these significant features of the framework can help users to determine the behaviors of any installed apps and start responding to actions to secure personal data. Despite the huge potential and many benefits that could be

gained from such research study, there are still many challenges and issues that need to be addressed mainly:

- Data flow control,
- Design a suitable user interface,
- Dynamic user privacy and data flow control policy,
- Novel lightweight privacy algorithms in the mobile technology side,
- User ability to modify the flow policy during runtime in response to incidents, and
- Modify application behavior that attempts to leak private data according to user decision.

To the best of our knowledge, the mobile Android community needs a solution that maintains the user controlling of permissions list, modifying them, monitor the data transmission, and change an App behavior based on configurable privacy and data flow policy.The authors would consider different discussed issues and challenges in their future work to design and implement a new approach to user privacy and data flow control for Android Apps.

References

[1] B. B. G. Malkiel and M. E. Stucke, "The Invisible Digital Threat: Mobile Ad Fraud 2019 Report," 2019. [Online]. Available: https://www.secure-d.io/mobileadfraud2019report/.

[2] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," in *Proceedings of the eighth symposium on usable privacy and security*, 2012, pp. 1–14, doi: 10.1145/2335356.2335360.

[3] I. I. Conference, C. Security, and C. Computing, "CUPA: A Configurable User Privacy Approach For Android Mobile Application," 2020.

[4] J. Seo, D. Kim, D. Cho, T. Kim, and I. Shin, "FLEXDROID: Enforcing In-App Privilege Separation in Android," in *NDSS*, 2017, no. February, pp. 21–24, doi: 10.14722/ndss.2016.23485.

[5] M. Hammad, H. Bagheri, and S. Malek, "The Journal of Systems and Software DelDroid: An automated approach for determination and enforcement of least-privilege architecture in android," *J. Syst. Softw.*, vol. 149, pp. 83–100, 2019, doi: 10.1016/j.jss.2018.11.049.

[6] G. Shrivastava and P. Kumar, "Privacy analysis of android applications: State-of-art and literary assessment," *Scalable Comput.*, vol. 18, no. 3, pp. 243–252, 2017, doi: 10.12694/scpe.v18i3.1304.

[7] Z. Alkindi, M. Sarrab, and N. Alzeidi, "Android Application Permission Model Issues and Privacy Violation," in *Free and Open Source Software Conference (FOSSC'2019-OMAN)*, 2019, no. April, pp. 47–51, [Online]. Available: https://www.researchgate.net/profile/Zainab_Alkindi/publication/332401070_Android_Application_Permission_Model_Issues_and_Privacy_Violation/links/5cb1d9cb92851c8d22e809b7/Android-Application-Permission-Model-Issues-and-Privacy-Violation.pdf.

[8] R. Neisse, G. Steri, D. Geneiatakis, and I. Nai Fovino, "A privacy enforcing framework for Android applications," *Comput. Secur.*, vol. 62, pp. 257–277, 2016, doi: 10.1016/j.cose.2016.07.005.

[9] Y. Shao, J. Ott, Q. A. Chen, Z. Qian, and Z. M. Mao, "Kratos: Discovering Inconsistent Security Policy Enforcement in the Android Framework," no. February, pp. 21–24, 2016.

[10] L. I. Jian, W. Zheng, W. Tao, T. Jinghao, Y. Yuguang, and Z. Yihua, "An Android Malware Detection System Based on Feature Fusion ∗," vol. 27, no. 6, 2018, doi: 10.1049/cje.2018.09.008.

[11] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012, no. 2, pp. 5–8, doi: http://www.internetsociety.org/hey-you-get-my-market-detecting-malicious-apps-official-and-alternative-android-markets.

[12] T. Bläsing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," *Proc. 5th IEEE Int. Conf. Malicious Unwanted Software, Malware 2010*, pp. 55–62, 2010, doi: 10.1109/MALWARE.2010.5665792.

[13] W. Enck *et al.*, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *Commun. ACM*, vol. 57, no. 3, pp. 99–106, 2014, doi: 10.1145/2494522.

[14] S. M. Kywe, Y. Li, J. Hong, and C. Yao, "Dissecting developer policy-violating apps: Characterization and detection," in *11th International Conference on Malicious and Unwanted Software, MALWARE 2016*, 2017, pp. 10–19, doi: 10.1109/MALWARE.2016.7888725.

[15] D. C. Nguyen, E. Derr, M. Backes, and S. Bugiel, "Short Text, Large Effect: Measuring the Impact of User Reviews on Android App Security & Privacy," *2019 IEEE Symp. Secur. Priv.*, pp. 555–569, doi: 10.1109/SP.2019.00012.

[16] D. S. Yadav and P. K. Doke, "Mobile Cloud Computing Issues and Solution Framework," *Int. Res. J. Eng. Technol.*, vol. 3, no. 11, pp. 1115–1118, 2016.

[17] Z. Xu and S. Zhu, "SemaDroid: A privacy-aware sensor management framework for smartphones," in *CODASPY 2015 - Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015, pp. 61–72, doi: 10.1145/2699026.2699114.

[18] G. Shrivastava, P. Kumar, D. Gupta, and J. J. P. C. Rodrigues, "Privacy issues of android application permissions: A literature review," *Trans. Emerg. Telecommun. Technol.*, no. September, pp. 1–17, 2019, doi: 10.1002/ett.3773.

[19] M. Hussain *et al.*, "Conceptual framework for the security of mobile health applications on Android platform," *Telemat. Informatics*, vol. 35, no. 5, pp. 1335–1354, 2018, doi: 10.1016/j.tele.2018.03.005.

[20] N. Asaddok and M. Ghazali, "Exploring the usability, security and privacy taxonomy for mobile health applications," in *International Conference on Research and Innovation in Information Systems, ICRIIS*, 2017, pp. 1–6, doi: 10.1109/ICRIIS.2017.8002472.

[21] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 5, pp. 1103–1112, 2017, doi: 10.1109/TIFS.2016.2646641.

[22] Z. Epstein, "WannaCry_ Everything you need to know about the global ransomware attack – BGR," http://bgr.com/tag/wannacry/, 2017.

[23] J. Patterson, "'Wanna Cry' virus infecting computers around the world, Tampa Bay area bracing for impact _ WFLA," 2017. http://wfla.com/2017/05/15/wanna-cry-virus-infecting-computers-around-the-world-tampa-bay-area-bracing-for-impact/.

[24] S. Bhandari, W. Ben, V. Jain, and V. Laxmi, "Android inter-app communication threats and detection techniques," *Comput. Secur.*, vol. 70, pp. 392–421, 2017, doi: 10.1016/j.cose.2017.07.002.

[25] D. Barrera, P. C. Van Oorschot, and A. Somayaji, "A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android," in *In Proceedings of the 17th ACM*

conference on Computer and communications security, 2010, no. 1, pp. 73–84, doi: 10.1145/1866307.1866317.

[26] Y. Xu, G. Wang, J. Ren, and Y. Zhang, "An adaptive and configurable protection framework against android privilege escalation threats," *Futur. Gener. Comput. Syst.*, vol. 92, pp. 210–224, 2019, doi: 10.1016/j.future.2018.09.042.

[27] "App permissions best practices |Android Developers," *Google Developer*, 2020. https://developer.android.com/training/permissions/usage-notes (accessed Aug. 25, 2020).

[28] G. Shrivastava and P. Kumar, "SensDroid: Analysis for Malicious Activity Risk of Android Application," *Multimed. Tools Appl.*, vol. 78, no. 24, pp. 35713–35731, 2019, doi: 10.1007/s11042-019-07899-1.

[29] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android Permissions Demystified," 2011.

[30] J. Song, C. Han, K. Wang, J. Zhao, and R. Ranjan, "An integrated static detection and analysis framework for android," *Pervasive Mob. Comput.*, vol. 32, pp. 15–25, 2016, doi: 10.1016/j.pmcj.2016.03.003.

[31] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission Re-Delegation: Attacks and Defenses," 2011.

[32] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A Whitebox Approach for Automated Security Testing of Android Applications on the Cloud," in *Proceedings of the 7th International Workshop on Automation of Software Test*, 2012, pp. 1–7.

[33] H. M. A. Maqsood, K. N. Qureshi, F. Bashir, and N. U. Islam, "Privacy Leakage through Exploitation of Vulnerable Inter-App Communication on Android," *2019 13th Int. Conf. Open Source Syst. Technol. ICOSST 2019 – Proc.*, pp. 31–36, 2019, doi: 10.1109/ICOSST48232.2019.9043935.

[34] A. H. Lashkari, A. F. Akadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," *Proc. – 2017 15th Annu. Conf. Privacy, Secur. Trust. PST 2017*, no. Cic, pp. 233–242, 2018, doi: 10.1109/PST.2017.00035.

[35] B. Kitchenham, "Procedures for Performing Systematic Reviews," 2004. doi: 1353–7776.

[36] Y. L. Phu H.Nguyen, Max Kramer, Jacques Klein, "An extensive systematic review on the model-driven development of secure systems," *Inf. Softw. Technol.*, vol. 68, pp. 62–81, 2015.

[37] V. S. Zlatko Stapiæ, Eva García López, Antonio García Cabot, Luis de Marcos Ortega, "Performing systematic literature review in software engineering," 2012.

[38] L. Li et al., "Static analysis of android apps: A systematic literature review," vol. 88, pp. 67–95, 2017, doi: 10.1016/j.infsof.2017.04.001.

[39] Statista, "Number of available applications in the Google Play Store from December 2009 to March 2017," 2017. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ (accessed Dec. 22, 2017).

[40] Polynomial, "Difference between information flow control, data flow integrity, and tainting – Information Security Stack Exchange," 2016. https://security.stackexchange.com/questions/125033/difference-between-information-flow-control-data-flow-integrity-and-tainting (accessed Aug. 31, 2020).

[41] X. Rong-na, L. Hui, S. Guo-zhen, G. Yun-chuan, N. Ben, and S. Mang, "Provenance-based data flow control mechanism for Internet of things," *Trans. Emerg. Telecommun. Technol.*, no. January, pp. 1–23, 2020, doi: 10.1002/ett.3934.

[42] D. Hedin and A. Sabelfeld, "A Perspective on Information-Flow Control," 2011.

[43] J. Bacon, D. Eyers, T. F. J. Pasquier, J. Singh, I. Papagiannis, and P. Pietzuch, "Information Flow Control for Secure Cloud Computing," *EEE Trans. Netw. Serv. Manag.*, vol. 11, no. 1, pp. 76–89, 2014.

[44] M. Sarrab, "Runtime Monitoring Using Policy Based Approach to Control Information Flow for Mobile Apps," *Int. J. Secur. Networks*, vol. 8, no. 4, pp. 212–230, 2013.

[45] S. M. Moura, "Floodgate: An Information Flow Control Platform for Distributed Mobile Applications Telecommunications and Informatics Engineering," 2015.

[46] C. Bae and S. Shin, "A collaborative approach on host and network level android malware detection," *Secur. Commun. Networks*, vol. 9, no. 18, pp. 5639–5650, 2016, doi: 10.1002/sec.1723.

[47] A. Tiwari, S. GroSS, and C. Hammer, "IIFA: Modular Inter-app Intent Information Flow Analysis of Android Applications," *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST*, vol. 305 LNICST, pp. 335–349, 2019, doi: 10.1007/978-3-030-37231-6_19.

[48] A. Bedford, "Enforcing Information-Flow Policies by Combining Static and Dynamic Analyses Enforcing Information-Flow Policies by," LAVAL, Canada, 2019.

[49] C. Hammer and S. Bugiel, "Secure Multi-Execution in Android," in *In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019, pp. 1934–1943.

[50] M. Backes, S. Bugiel, E. Derr, S. Gerling, and C. Hammer, "R-Droid: Leveraging Android App Analysis with Static Slice Optimization Invited Paper," pp. 129–140, 2016.

[51] R. Liu, J. Cao, K. Zhang, W. Gao, J. Liang, and L. Yang, "When Privacy Meets Usability: Unobtrusive Privacy Permission Recommendation System for Mobile Apps Based on Crowdsourcing," *IEEE Trans. Serv. Comput.*, vol. 11, no. 5, pp. 864–878, 2018, doi: 10.1109/TSC.2016.2605089.

[52] D. Geneiatakis, I. Nai, I. Kounelis, and P. Stirparo, "A Permission Verification Approach for Android Mobile Applications," *Comput. Secur.*, vol. 49, pp. 192–205, 2015.

[53] J. Tang, R. Li, H. Han, H. Zhang, and X. Gu, "Detecting permission over-claim of android applications with static and semantic analysis approach," in *Proceedings – 16th IEEE International Conference on Trust, Security, and Privacy in Computing and Communications, 11th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems*, 2017, pp. 706–713, doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.303.

[54] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy, and I. Hegazy, "AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach," *Information*, vol. 10, no. 10, p. 326, 2019, doi: 10.3390/info10100326.

[55] L. Li, A. Bartel, F. Bissyand, J. Klein, and Y. Le Traon, "ApkCombiner: Combining Multiple Android Apps to Support Inter-App Analysis," in *IFIP International Information Security Conference*, no. Icc, pp. 513–527.

[56] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic," *Proc. 14th Annu. Int. Conf. Mob. Syst. Appl. Serv. (MobiSys'16)*, pp. 361–374, 2016, doi: 10.1145/2906388.2906392.

[57] S. Arzt et al., "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps."

[58] S. Zimmeck et al., "Automated Analysis of Privacy Requirements for Mobile Apps," in *2016 AAAI Fall Symposium Series*, 2016, vol. 3078, no. 132, doi: 10.14722/ndss.2017.23034.

[59] B. P. S. Rocha, M. Conti, S. Etalle, and B. Crispo, "Hybrid Static-Runtime Information Flow and Declassi fi cation Enforcement," *IEEE Trans. Inf. forensics Secur.*, vol. 8, no. 8, pp. 1294–1305, 2013.

[60] J. Reardon, U. C. Berkeley, S. Egelman, and U. C. B. Icsi, "50 Ways to Leak Your Data: An Exploration of Apps ' Circumvention of the Android Permissions System," in *In 28th Security Symposium (USENIX Security 19)*, 2019, pp. 603–620.

[61] A. Sadeghi, R. Jabbarvand, N. Ghorbani, H. Bagheri, and S. Malek, "A temporal permission analysis and enforcement framework for Android," in *Proceedings – International Conference on Software Engineering*, 2018, pp. 846–857, doi: 10.1145/3180155.3180172.

[62] A. Alzaidi, S. Alshehri, and S. M. Buhari, "DroidRista: a highly precise static data flow analysis framework for Android applications," pp. 523–536, 2020.

[63] F. Liu, H. Cai, G. Wang, D. D. Yao, K. O. Elish, and B. G. Ryder, "MR-Droid: A Scalable and Prioritized Analysis of Inter-App Communication Risks," 2017.

[64] B. Liu *et al.*, "Follow My Recommendations: A Personalized Privacy Assistant for Mobile App Permissions This paper is included in the Proceedings of the Follow My Recommendations: A Personalized Privacy Assistant for Mobile App Permissions," no. Soups, 2016.

[65] P. Wijesekera, A. Baokar, L. Tsai, and J. Reardon, "The Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences," in *In 2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 1077–1093.

[66] Y. Cao, Y. Fratantonio, A. Bianchi, and M. Egele, "EdgeMiner: Automatically Detecting Implicit Control Flow Transitions through the Android Framework," no. February, pp. 8–11, 2015.

[67] L. Li et al., "IccTA: Detecting Inter-Component Privacy Leaks in Android Apps."

[68] A. B. Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden and and P. M. Jacques Klein, Yves Le Traon, Damien Octeau, "Flowdroid: Precise context, ow, eld, object-sensitive and lifecycle-aware taint analysis for android apps.," *Program. Lang. Des. Implement.*, vol. 46, no. 6, pp. 259–269, 2014.

[69] F. Wei, S. Roy, and X. Ou, "Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps Categories and Subject Descriptors," in *Proceedings of*

the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 1329–1341.

[70] X. Wei, "ProfileDroid: Multi-layer Profiling of Android Applications Categories and Subject Descriptors."

[71] S. V. Sbirlea D, Burke MG, Guarnieri S, Pistoia M, "Automatic detection of inter-application permission leaks in android applications," *IBM J Res Dev*, vol. 57, no. 6, pp. 1–12, 2013.

[72] J. Kim, Y. Yoon, and K. Yi, "S CAN D AL: Automated Security Certification of Android Applications."

[73] D. Schreckling, D.- Passau, J. Posegga, D.- Passau, and D. Hausknecht, "Constroid: Data-Centric Access Control for Android," in *In Proceedings of the 27th ACM Symposium on Applied Computing (SAC)*, 2012, pp. 1478–1485.

[74] "TrustDroid TM": Preventing the use of SmartPhones for information leaking in corporate networks through the use of static analysis taint tracking Zhibo Zhao and Fernando C. Colon Osorio 2. Overview of the Android environment," no. March 1999, pp. 1–9, 2007.

[75] E. Chin, A. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in Android," in *In Proceedings of the 9th international conference on Mobile systems, applications, and services*, 2011, pp. 239–252, doi: 10.1145/1999995.2000018.

[76] P. Gilbert and L. P. Cox, "Vision: Automated Security Validation of Mobile Apps at App Markets."

[77] B. Rashidi, C. Fung, and T. Vu, "Android fine-grained permission control system with real-time expert recommendations," *Pervasive Mob. Comput.*, vol. 32, pp. 62–77, 2016, doi: 10.1016/j.pmcj.2016.04.013.

[78] Z. Safavi, S., and Shukur, "CenterYou: A cloud-based Approach to Simplify Android Privacy Management," 2020.

[79] G. L. Scoccia, M. Autili, and P. Inverardi, "A self-configuring and adaptive privacy-aware permission system for Android apps," in *Proceedings – 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020*, 2020, pp. 38–47, doi: 10.1109/ACSOS49614.2020.00024.

[80] G. L. Scoccia, I. Malavolta, M. Autili, A. Di Salle, and P. Inverardi, "Enhancing Trustability of Android Applications via User-Centric Flexible Permissions," *IEEE Trans. Softw. Eng.*, vol. PP, no. X, pp. 1–1, 2019, doi: 10.1109/the.2019.2941936.

[81] M. Diamantaris, E. P. Papadopoulos, and J. Polakis, "REAPER: Real-time App Analysis for Augmenting the Android Permission System,"

in *In Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, 2019, pp. 37–48.

[82] M. Y. Wong and D. Lie, "IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware," in *Proceedings of the annual symposium on the network and distributed system security (NDSS)*, 2016, no. February, pp. 21–24.

[83] R. Schuster and E. Tromer, "DroidDisintegrator: Intra-Application Information Flow Control in Android Apps," *ASIA CCS '16 Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, pp. 401–412, 2016, doi: 10.1145/2897845.2897888.

[84] J. Gu, Y. Calvin, H. Xu, C. Zhang, and H. Ling, "Privacy concerns for mobile app download: An elaboration likelihood model perspective," *Decis. Support Syst.*, vol. 94, pp. 19–28, 2017, doi: 10.1016/j.dss.2016.10.002.

[85] G. Suarez-tangil, J. E. Tapiador, P. Peris-lopez, and A. Ribagorda, "Evolution, Detection, and Analysis of Malware for Smart Devices," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 2, pp. 961–987, 2013.

[86] P. Hornyack and S. Schechter, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," *in Proceedings of CCS*, 2011, pp. 639–651.

[87] Acpm, "Inspeckage: Android Package Inspector - dynamic analysis with API hooks, start unexported activities and more," *acpm*, 2017. https://github.com/ac-pm/Inspeckage (accessed Sep. 17, 2020).

[88] K. Cotterell, I. Welch, and A. Chen, "An Android Security Policy Enforcement Tool," *in INTL journal of electronics and telecommunications*, 2015, vol. 61, no. 4, pp. 311–320, doi: 10.1515/delete-2015-0040.

[89] Y. Zhang, M. Yang, Z. Yang, G. Gu, P. Ning, and B. Zang, "Permission use analysis for vetting undesirable behaviors in android apps," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 11, pp. 1828–1842, 2014, doi: 10.1109/TIFS.2014.2347206.

[90] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: Automatic security analysis of smartphone applications," in *CODASPY 2013 – Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, 2013, pp. 209–220, doi: 10.1145/2435349.2435379.

[91] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "AppIntent: analyzing sensitive data transmission in android for privacy leakage detection," in *Proceedings of the 2013 ACM SIGSAC*

conference on Computer & communications security - CCS '13, 2013, pp. 1043–1054, doi: 10.1145/2508859.2516676.

[92] H. Lee, D. Kim, M. Park, and S. Cho, "Protecting data on the Android platform against privilege escalation attack," *Int. J. Comput. Math.*, vol. 93, no. 2, pp. 401–414, 2016, doi: 10.1080/00207160.2014.986113.

[93] M. Zhang, H. Yin, and A. App, "Transforming and Taming Privacy-Breaching Android Applications," no. February, pp. 7–8, 2012.

[94] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints."

[95] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing Smartphone Applications (on Android)," in *International Conference on Trust and trustworthy computing*, 2011, pp. 93–107.

[96] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "MockDroid: Trading privacy for application functionality on smartphones," in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications – HotMobile'11*, 2011, no. February, p. 49, doi: 10.1145/2184489.2184500.

[97] A. Fuchs, A. Chaudhuri, and J. Foster, "CRePE:context-related policy enforcement for android," in *Proceedings of the 13th international conference on Information security, ser. ISC10. Berlin, Heidelberg: Springer-Verlag*, 2011, pp. 331–345, doi: 10.1.1.164.6899.

[98] A. P. Fuchs, A. Chaudhuri, and J. Foster, "SCanDroid: Automated Security Certification of Android Applications," *Read*, vol. 10, no. November, p. 328, 2010, doi: 10.1.1.164.6899.

[99] Y. J. Park, D. Chung, K. Kim, and J. Kim, "An Enhanced Security Policy Framework for Android Made Harta Dwijaksara," 2011.

[100] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, "Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem," *IEEE Trans. Mob. Comput.*, vol. PP, no. c, p. 1, 2019, doi: 10.1109/TMC.2019.2903186.

[101] A. Security, "Mobile App Security and Privacy Analysis," *Ostorlab*, 2017. Ostorlab,.co.

[102] Q. Qian, J. Cai, M. Xie, and R. Zhang, "Malicious behavior analysis for android applications," *Int. J. Netw. Secur.*, vol. 18, no. 1, pp. 182–192, 2016.

[103] P. Singh, P. Tiwari, and S. Singh, "Analysis of Malicious Behavior of Android Apps," *Procedia - Procedia Comput. Sci.*, vol. 79, pp. 215–220, 2016, doi: 10.1016/j.procs.2016.03.028.
[104] T. Oluwafemi and O. Riva, "Per-App Profiles with AppFork: The Security of Two Phones with the Convenience of One," *Microsoft*, 2014.
[105] T. Oluwafemi, "Using Component Isolation to Increase Trust in Mobile Devices," 2015.
[106] S. Lortz, D. Schneider, and A. Weber, "Cassandra: Towards a Certifying App Store for Android," in *In Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, 2014, pp. 93–104.
[107] S. Holla and M. M. Katti, "Android Based Mobile Application Development and its Security," *Int. J. Comput. Trends Technol.*, vol. 3, no. 3, pp. 486–490, 2012.
[108] T. Mohini, S. A. Kumar, and G. Nitesh, "Review on Android and Smartphone Security," *Int. J. Eng. Sci.*, vol. 1, no. 6, pp. 12–19, 2013.
[109] C. Mulliner, G. Vigna, D. Dagon, and W. Lee, "Using Labeling to Prevent Cross-Service Attacks Against Smart Phones," pp. 91–108, 2006.

Biographies

Zainab Rashid Alkindi is a Ph.D. student at Sultan Qaboos University. Zainab has accomplished many achievements throughout her academic life. She has conducted many types of research in IoT, Network, and Security & user privacy, and she has been participating in different conferences. She worked as a research assistant in the Communication and Information Research Center (CIRC) for two years, which shaped her research skills.

She obtained her MSc in the Networking area in 2017 from Sultan Qaboos University.

Mohamed Sarrab is currently working as a researcher and deputy director of the Communication and Information Research Center (CIRC), Sultan Qaboos University (Muscat, Sultanate of Oman). He obtained his Ph.D. in Computer Science from De Montfort University (UK). His research interests are in the areas of, Software Engineering, Mobile learning (M-learning), and Mobile Cloud Computing. He is also interested in mobile application security, in particular, Access Control and Policy-Based System Management, Runtime Verification and Information Flow Control, and viz. Software Systems, where security requirements are managed using loosely coupled components that enforce high-level security requirements.. He is a senior member of the IEEE, the IEEE Computer Society, and IEEE Communications Society.

Nasser Alzeidi received his Ph.D. degree in Computer Science from the University of Glasgow (UK) in 2007. He is currently an Associate Professor

of computer science and the director of the Center for Information Systems at Sultan Qaboos University, Oman. His research interests include performance evaluation of communication systems, wireless networks, interconnection networks, System on Chip architectures, and parallel and distributed computing. He is a member of the IEEE and the Chair of the IEEE Computer Society Chapter in Oman.