Vol. 6, No. 2 | July – December 2022

Collusion Detection using Predictive Functions based on Android Applications

Asad Hameed Soomro[1]*, Samina Rajper[1], Aurangzeb Magsi[1*], Samar Abbas Mangi[1], Aneela Jan Soomro[2]

Abstract:
Android is used by most of the population of the users. It is an attractive target for malicious application developers due to its open-source nature. A number of applications are developing day by day for android devices to serve the purpose of data stealing activity. A collision attack is one of the types of applications or programs used for data stealing from android devices. During this attack, different apps communicate via Inter-Process Communication (IPC) for a variety of purposes. In this paper, a dynamic approach is proposed for automatic collision detection between communications among different applications. The focus of the study is on the sharing of multiple data types. Moreover, to select applications for analysis is a difficult task to perform and two predictive functions have been used in this regard. The evaluation is performed on a dataset of 800 android applications for analyzing the colluding couples. The developed methodology produces an accuracy of 97.2% during the experiments by the developed system.

Keywords: *Security, Android, Collision Detection, Formal Model, Predictive function*

1. Introduction

According to surveys [1, 2], Android is an open-source operating system that is dominating the mobile market and ranks first among mobile operating systems. There are massive numbers of apps that are rapidly growing. According to [3], there were over 3.3 million apps accessible in Google Play during the first half of 2018, with millions more available in unauthorized stores. Despite this, the Apple app stores secure 2nd place among mobile operating systems. Along with applications there is a remarkable growth for android operating system has also been reported. It has been reported that most of the applications compromise the users' confidential data and stole or damage the data. These are called malicious applications. Likewise, most of the applications are over-privileged and ask for a lot of permissions without any need for permission for the execution of the application. These over-privileged apps direct or indirectly access the users' data [4] and spread them into the sinks (external sources).

For application environment implementation, Android supports mobile devices. It consists of the operating system, the application charter, and the fundamental functionality. The Android operating system is built on the Linux kernel [5], which is used for device drivers, memory management, process monitoring, and networking. The next stage comprises of Android built-in archives. These archives are used by most of the higher-layer libraries, which are developed in C/C++. Integrating these archives in Android applications is performed through Java built-in interface. Run-time is an additional level, covering the Dalvik simulated machine and the

[1]Department of Computer Science, Shah Abdul Latif University, Khairpur Mirs, Pakistan
[2]Computer Department, Government College for Women Khairpur Mirs, Pakistan
Corresponding Authors: aurangzeb.magsi.sef@gmail.com, asad.soomro31@yahoo.com

primary archives. Dalvik runs .dex libraries are considered to be further dense and memory-efficient than Java class libraries. Through java created primary libraries, Android-explicit archives, and Java 5 SE suites give a large subset. The Google-provided apparatus, as well as trademarked extensions or services, are included in the app structure layer, which was written in Java. For installation, each software is packaged in an.apk archive.

This package is similar to a standard Java jar file in that it contains all of the app's non-code and code attributes (such as images or primary files). The Android software development kit (SDK) provides APIs for Android applications that are written in Java. Applications like phones, browsers, email clientst, and more are provided by the highest application level. Android also implements a permission mechanism for the security of the application. Every application should ask for permission that can be required in the Android manifest file during the coding phase. During the first execution of the app after installation these permissions will be granted by the user in order to use the full functionality of the application.

This permission mechanism becomes ineffective when more and more permission APIs are executed. This has raised an alarming issue because malicious writers write code to perform specious actions when these permission mechanisms become ineffective to steal the sensitive data of the user [6, 7, 8].

With respect to this context, Collision attack [9] which is called a new style has been characterized by vicious developers. This type of attack is characterized in such a manner that the malicious actions are divided into several apps that can be executed, for which the app requires least permissions [10]. The existing antimalware apps are unable to identify such type of attack vectors because of its distributed malicious payload [11]. This scenario can be very clear in the statement: the first application required permission to read private data for providing it to the second app, which spread it to the sinks (outsources). In this manner, only read permission is required by the first application and Internet access permission is required by the second app to initiate attack [10]. These colluding apps if analyzed individually will not be identified by antimalware to find malicious code, since the impact will be performed by their collision [12].

Most of the time applications are not independent from each other in the Android mechanism, the applications are connected through Inter-Component Communication (ICC). This functionality is provided by Android to their developers to lighten the coding burden and computing cost in the applications by allowing inter-application cooperation to exchange information among components that may be of same application or different applications [13]. On the other hand, this functionality can also raise the issue that this mechanism is being misused by malicious developers to perform malicious actions for different to steal users' sensitive data for different purposes [14].

By following these concepts, this paper presents a tool for implementing a methodology that analyzes Android applications to detect collusion among apps by using a new predictive function that is able to minimize the number of the examined apps. To define the function μ-calculus temporal logic is used. The focus of the paper is on multi-valued resources shared exploiting Android SharedResources

The following is the paper's structure. Section 2 offers a literature overview of prior works, section 3 outlines the problem, section 4 defines the suggested method for collusion detection in Android applications, and section 5 concludes with future research directions.

2. Literature Review

As an open-source platform, Android has a stronghold on the mobile market, which has seen rapid expansion in recent years. The number of apps available to users has exploded as well. In comparison to other operating systems, [15] reveals that Google's Android Operating System holds up to 91 percent of the market. Furthermore, it offers a significant effect value for Android-based products. The

Android operating system protects users' privacy by implementing a permission-based approach that limits all apps' access to a user's personal information. Each app requires a set of rights, which the app developer determines and the user accepts during the app installation process [7, 16]. However, these permission systems have become useless when malicious applications are used to steal the sensitive data of the user.

Different studies regarding android malicious applications are reported. [6, 14] Performed surveys in order to analyze the footprints that lead to the collection of sensitive data and to analyze the android security approaches, respectively. Likewise, a study was conducted in such a way to propose a threat model that enlightens the existing threats of the android operating system for collecting sensitive resources [17]. In this manner, an experiment was performed for classification and feature extraction using machine learning algorithms. Similarly, two asymmetric methodologies to spot malicious samples in android have been proposed in [18] based on machine learning and model checking. Evolution shows effective results by both design methods and finds HummingBad (a malware family) footprints in malicious android applications. Moreover, a study to measure the security-related issues was conducted in [9] on the permission mechanism of android. In this connection, defense mechanism model name PBAD (Permission Based attack defense) was proposed. The proposed model first examines the application interacting with each other in order to perform malicious actions then-after, shelter the permission-protected interfaces on the innocuous applications. Another study was reported that uses a runtime instrumented testing environment that executes thousands of android apps [7]. During testing the apps' behavior for transmitting sensitive data over the network by using unauthorized access was analyzed. Then, reverse engineering was used to reverse the activities of apps and to determine the malicious actions evidence.

In the light of the above research, a study was performed for measuring the effectiveness of using the Android permission mechanism. The novelty of the study was to analyze the relationship between permission mechanisms and users' private data. Additionally, repeated permissions were analyzed in the android application, and users' perspective to understand the permissions was also discussed. Additionally, some studies in users' contexts have also been reported [19, 20]. A user-centric approach has been proposed in this regard that allows end-users to customize the requested app permissions on a per-feature basis and some newly acquired responses regarding users' privacy settings that affect the functionality of applications in contrast with previous studies that were on the spotlight, respectively.

Despite of all these different ways for specious activities performed by malicious app developers to steal sensitive data of the users, a new footprints of a method have also been found for performing spurious actions. This method is called Inter-Component Communication (ICC) / Inter-app Communication (IAP) which executes collision attacks and communication between applications.

Followed by this statement, a static approach for ICC/IAC was proposed in [21] to analyze the flow of information from source to sink. The study was based on android intent sharing between different applications for sharing of sensitive data. Presented tactic was claimed as first automatic information flow analysis of ICC/IAC that uses short summaries instead of analyzing all tuples of apps. A similar study was also performed to tackle the security issue of android that uses reflective methods [22]. The study was based on the static examination of apps that communicate for unauthentic purposes and perform specific tasks. In this connection, an automated tool called DroidRA was proposed that shows effective statistical analysis on the dataset during evolution for threat detection.

Likewise, a study to observe the exploitability of prospective threats by analyzing of implicit information flow (IIF) has been discussed [23]. In this regard, switch-transfer-based semantic analysis were applied.

The result was effective and efficient using proof-of-concepts for transmitting sensitive data by avoiding state-of-the-art privacy monitors. The study was summarized by proposing a solution for defending against IIF leveraging a special control dependence tracking technique. Moreover, a permission-centered study for secure information flow was also performed [24]. The author proposed an interpretation algorithm for primary security type systems. The soundness of the study was based on a feature that restricts that branching brought by consent testing and allows more specific security plans to be imposed. A Mutation based analysis was also performed by the MUTAFLOW prototype in [25]. The prototype track the mutation changes dynamically returned from sensitive resources and is determined to outsource where data was received. Then-after, the flow between the source and destination was mechanized. The prototype shows more effective results than other existing tools and mechanisms.

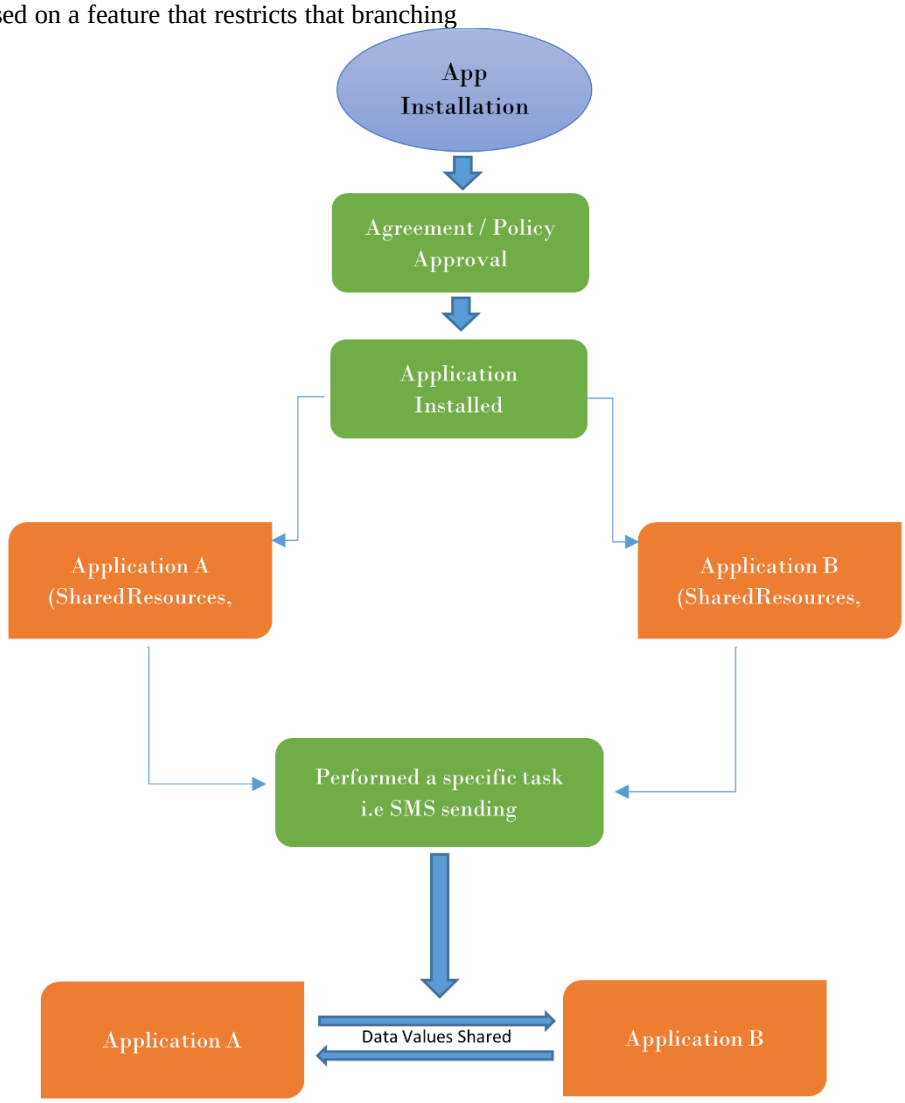

Fig. 1. Collision between Applications using SharedResources

Application collusion studies are performed in variety of ways by other researchers that uses the same attack vector which non-other than collusion and detection of these applications by using different methods are increasing the accuracy but there are a variety of gaps that still exists. The current studies mostly performed static approaches in order to analyze the application which is insufficient. Moreover, analyzing individual applications is not enough to find the communication channel between different applications. In this paper, a dynamic approach has been proposed that is able to find the coupling or communication channel between applications.

3. Problem Statement

As "Fig 1" depicts the behavior of two different applications A and B. After installation and acceptance of policies, the application is installed in the device as per the desire of the end-user. After installation, the individual installed applications can transmit data through the same SharedResources name i.e "Shared". Subsequently, a hidden door will be opened between applications for coupling or communication purposes and at that point of time multiple data values will be shared. The next statement will make it clear: Two apps can make a couple as App A sense the private data from the users' device that could be transmitted to App B, which can be spread to sinks. For this kind of action, the first app-only entails consent to read the data, the other one only needs the usage of an Internet connection [10]. Because commercial antimalware solutions examine all apps individually, they will not be able to detect the threat or find the mistakes, because the damage will be created by their collusion [12]. In this study, a tool that implements an approach for detecting colluding android applications using predictive functions is proposed. Precisely, the focus is on multiple data type values exploiting Android SharedResources.

4. Proposed Methodology

Initially, the stage entails defining a formal model that will be utilized to create an android application model. As a result, a general model is defined that can verify all types of properties on the system.

Starting from the bytecode, the formal model is generated using the Calculus of Communicating Systems (CCS) approach, which replicates the actions of an application. CSS is defined as follows using the Backus Normal Form (BNF):

P ::=0 | a.P1 | A | P1 + P2 | P1| P2 | P1[b/a] | P1\a

In the order listed above, the pieces of the syntax are:

Inactive process:

The process a.P1 can perform an action a and continue as the process P1.

Process identifier:

Write $A \stackrel{def}{=} P\_1$ to refer to the process P1 with the identifier A. (which may contain the identifier A itself, i.e., recursive definitions are allowed).

Choice:

The process P1 + P2 can be carried out as either process P1 or the process P2.

Parallel composition:

P1| P2 indicates that processes P1 and P2 are active at the same time.

Renaming:

P1[b/a] is the process P1 with all a-named actions renamed to b.

Restriction:

P1\a is the process P1 without action a.

An Android app, commonly known as an.apk (Android Package), is a modified version of the popular.jar archive file. This sort of file contains the Dalvik Virtual Machine's executable code, such as the .dex file, the supply folder (icons, graphics, and sounds), and the Manifest file. The.apk file, which could be the first step, can be used to obtain bytecode information. The following are the steps that will be followed to obtain Javabytecode via an apk file:

- Mining the java class file and the handbooks from the.jar file using the Java Archive Tool utility2;
- Generation of the.jar file from the.apk file using a device named dex2jar1;
- Creation of the Bytecode Android application that summons the BCEL (Byte Code Engineering Library3).

Following the acquisition of Java Bytecode, the researchers created an interpretation algorithm (see Mercaldo et al., 2016). For each Java Bytecode line, the algorithm can generate a CSS process. The procedure then encrypts the coding and characterizes the opcodes by encrypting the steps it takes (i.e. the control flow among multiple instructions).

4.1 Predictive Functions: PUT and GET

Detecting the colluding apps is a very difficult task to perform because the existing tools present in the market are ineffective and unable to find the coupling apps. There is a huge amount of applications available in the official and unofficial markets for the users and this is the reason for the exponential growth of examination costs.

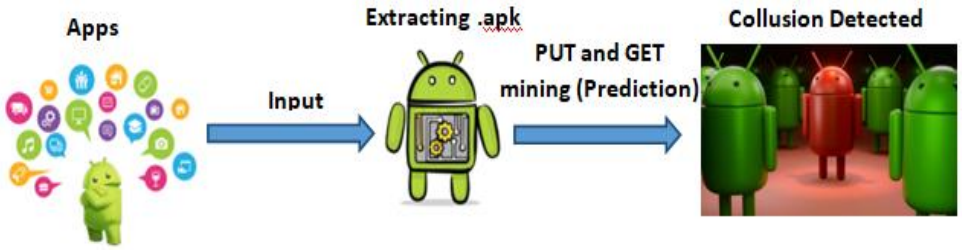

Fig. 2. Proposed Methodology

Consider the number of apps n, to evaluate n number of applications the system needs to perform n2 tests, and n3 tests will be performed for all the possible triples and so on "Fig 2". In this context, there is a solid need for a method that reduces the search space from the large dataset for colluding candidates and analyzes the group of apps that can be selected for collision detection.

The study presents two predictive functions: Put and Get

The Put focused on the SharedResources and analyzed each possible code route for every read/write operation on an integer, string, or float share resource. It will divide the apps into groups based on how they use shared resources.

Let's take a look at the Android code sample below to see how SharedResources work. It is an example of SharedResources invocation; specifically, the code snippet receives an integer value from the SharedResources by using the getInt methods (stored in the value1 variable).

The function of SharedResources can be demonstrated by considering the piece of Android code shown below: The invocation of SharedResources has been mentioned as an example, specifically, the piece of code is invoked using getInt methods an Integer value from the SharedResources (stored in the sharing1 variable

1. SharedResources sharedResources = this.

getSharedResources (SharedResources,

Context. WRITEABLE);

2. Int value1 = sharedResources.getInt(sharing1

,defaultValue);

The second Android piece of code depicts the SharedResources writing invocations.

1. SharedResources sharedResources = this.

getSharedResources(SharedResources, Context

2. WRITEABLE);

3 SharedResources.Editor editor = sharedResources.

edit();

4 editor.putInt (sharing1, "010");

More precisely, the example shows that the particulars of getInt method is kept in the SharedResources. As mentioned in literature [12], it is an easy task for multiple application to share the private record only by acknowledging the name of the SharedResources (in the code snippets SharedResources)

In general, two separate apps can perform different operations on a shared resource (i.e get and put actions).

To encode these actions, we use the μ-calculus temporal logic, which states that each proposition and variable is a formula, such as if Φ is a formula and is a formula, then [a] Φ. is a formula as well.

- If a process is capable of performing the following sequence of activities, it can perform a "put" on a shared resource (Table 1 – Formula 1): callgetSharedResources references, calledit, callputInteger, callcommit;
- An app can do a "get" on a shared resource if the process can perform the following sequence of actions: callgetSharedresources, callgetInteger (Table 1 – Formula 2).

In the proposed methodology predictive functions 'Put' and 'Get' are defined to detect a couple of collusion applications in a short time. These predictive functions will be able to find the two multiple sets of applications which will be examined. The first one will probably that validate put property and the second one will validate the get property.

Furthermore, the predictive functions will reduce computing capacity by checking a temporal logic formula in the CCS procedures displaying the applications. The infinite sequence behavior N0[Q0> N1[Q1> … N the following infinite sequence behavior σ = ≪ N0 … Nn, Nn, … >> is obtained. Let a and b be two arbitrary temporal formulas; p be many place; t be a transition; q, q1 …, qn be rigid variables; σ = ≪ N0, N1, … >> be a behavior; and σl = ≪ NL, Ns+1, … >> be a L-step-shifted behavior sequence. We define the semantics of temporal formulas recursively as follows:

$(1)\sigma[\![\rho(x_{1,\ldots\ldots x_n}]\!] \equiv M_0\,[\![\rho(x_{1,\ldots\ldots x_n}]\!]$

$(2)\sigma[\![t]\!] \equiv M_0\,[\![t]\!]M_1$

$(3)\sigma[\![\neg u]\!] \equiv \neg\sigma[\![u]\!]$

$(4)\sigma[\![u\ \Lambda\ v]\!] \equiv \sigma[\![u]\!]\ \Lambda\ \sigma[\![v]\!]$

$(5)\sigma[\![\forall x u]\!] \equiv \forall x\ \sigma[\![u]\!]$

$(6)\sigma[\![u]\!] \equiv \forall x \in \mathbf{Nat}\ \sigma^n[\![u]\!]$

The work flow of Get and Put property is subsequently one after the other minimizes the searching capacity of the applications used for collusion based on execution flow.

The proposed methodology will have the ability to find all the sets of colluding applications that shows the footprints of threat using SharedResources.

5. Evaluation and Results

To test the proposed methodology, a dataset is generated from different official and un-official sources of android applications. The dataset contains 800 android applications for experimental purpose. The dataset is evaluated in three different viewpoints:

In the first perspective applications are analyzed to find out the colluding couples which perform collision for sharing the data using SharedResources. Hence by applying our proposed methodology, it has been found that 60 out of 120 applications use SharedResources method to share the data between applications and other sinks. "Fig 3"

is the graphical representation of colluding evaluation.

The second perspective to analyze the dataset is to identify the type of data which is transmitted among these couples. The Get and Put functions produced effective results in this regard. "Fig 4" shows that 38% of the applications transmit the string type data using SharedResources. In addition, integer and float type data are also transmitted with 32% and 30%, respectively.

After analyzing the data type, the size of the data transferred by colluding applications also analyzed. "Fig 5" describes that 37% of the colluding couple transmits 301-500KB data between them, which is the highest size calculated from the dataset. Likewise, the minimum size of the transmitted data noted as 1KB.

TABLE I. FORMULA DESCRIBING THE PREDICTIVE FUNCTIONS GET AND PUT

Formula 1

$$\phi PUT = \mu Y.\ (callgetSharedResources)\ \phi PUTa\ W(-callgetSharedResources)$$

$$\phi PUTa = \mu Y.\ (calledit)\ \phi PUTb\ W\ (-Calledit)\ Y$$

$$\phi PUTb = \mu Y.\ (callputInt)\ \phi PUTb\ W\ (-callputInt)Y = \mu Y.\ (callcommit)\ ss\ Y\ (-Callcommit)\ Y$$

Formula 2

$$\phi GETa = \mu Y.\ (callgetSharedResources)\ \phi GETa\ W\ (-callgetSharedResources)\ Y$$

$$\phi GETb = \mu Y.\ (callgetInt)\ ss\ \vee\ (-callgetInt)\ Y$$

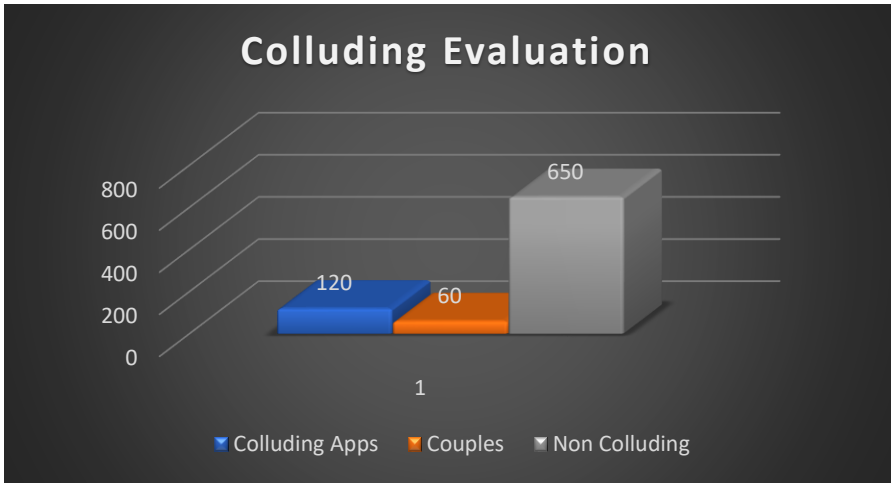

Fig. 3. Colluding Evaluation

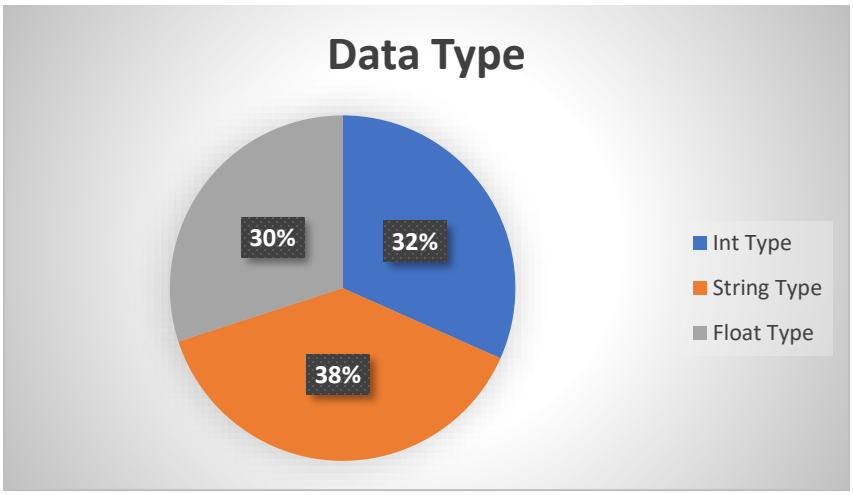

Fig. 4. Data Type Transmitted between Colluding Applications

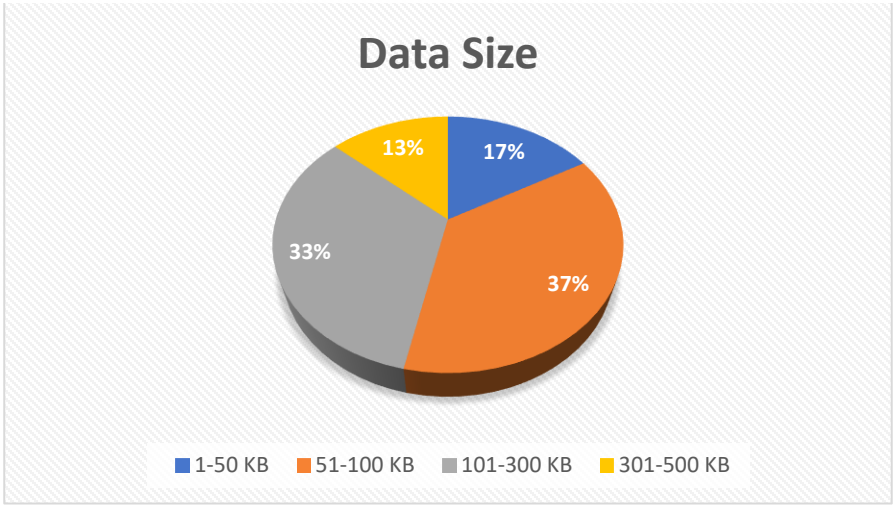

Fig. 5. Estimated Data Size Transmitted between Colluding Applications

Furthermore, a confusion table is selected to represent the findings because the system generated genuine positive and false negative results for the provided input, as well as true negative and false positive outcomes at times. As shown in "Fig 5.2", Table 2 is the further demonstration of evaluation of the dataset to identify string type data based on the computed size. The developed approach successfully recognized 647 true positive and 111 false negative applications. The System also produced some unexpected outcomes, such as false positives and true negatives.

TABLE II. IDENTIFICATION OF STRING TYPE DATA ACCORDING TO THE CALCULATED SIZE

Total Apps = 800
1 KB – 50 KB = 351
51 KB – 200 KB = 221
201 KB – 500 KB = 228

Data Size (String)	True Positive	False Positive	True Negative	False Negative
1 KB – 50 KB	303	2	0	46
51 KB – 200 KB	180	4	13	24
201 KB – 500 KB	164	6	17	41

Table 3, on the other hand, shows the generated results for integer-type data based on the calculated size. The technology successfully detected 705 true positive applications and 82 false negative applications. The data of the float type also examined according to the size, as shown in table 4. The proposed approach is also successful in identifying float type data based on its size. Seven hundred and eighty-five applications yielded 705 and 81 true positive and false negative results, respectively, with a minor number of false positives and true negatives.

TABLE III. IDENTIFICATION OF INTEGER TYPE DATA ACCORDING TO THE CALCULATED SIZE

Total Apps = 800
1 KB – 50 KB = 298
51 KB – 200 KB = 309
201 KB – 500 KB = 193

Data Size (Integer)	True Positive	False Positive	True Negative	False Negative
1 KB – 50 KB	263	4	1	30
51 KB – 200 KB	287	2	0	20
201 KB – 500 KB	155	4	2	32

TABLE IV. IDENTIFICATION OF FLOAT TYPE DATA ACCORDING TO THE CALCULATED SIZE

Total Apps = 800
1 KB – 50 KB = 193
51 KB – 200 KB = 309
201 KB – 500 KB = 298

Data Size (Float)	True Positive	False Positive	True Negative	False Negative
1 KB – 50 KB	156	3	2	32
51 KB – 200 KB	288	2	0	19
201 KB – 500 KB	263	4	1	30

The proposed methodology produced effective results to identify all 60 colluding couples among datasets which perform spiteful actions utilizing the SharedResources. Furthermore, cumulative accuracy of 97.2% has been achieved during the experimental study

6. Conclusion and Future Work

Android permission mechanism is used to secure the users' devices from threats or vulnerabilities but this permission mechanism becomes useless when more and more permissions are being executed. This paper highlights a new trend in this context which is called collision attack. This type of attack is used by most of the malicious applications to share data between them. We present a novel method that is beneficial to identify application couples that perform collision attacks. The proposed method uses predictive functions to analyze the dataset of applications. These functions are useful for reducing the computing cost and search space. The system produces effective results for the colluding detection method. The overall accuracy of the system is 97.2% for the identification of colluding applications and for the detection of the type and size of the data shared by these colluding couples using SharedResources. For future work more applications will be analyzed for the detection of sharing of some other type of data like image, audio, video. In addition, a method will also be proposed to prevent the application for sharing of data.

REFERENCES

[1] Statista 2021, 'App stores_ number of apps in leading app stores 2021 _ Statista', Statista 2021, 2021. [Online]. Available: <https://www.statista.com/statistics/266136/. [Accessed: 05-February-2021]

[2] Enck, William, et al. "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones." ACM Transactions on Computer Systems (TOCS) 32.2 (2014): 1-29.

[3] Statista 2021, 'App stores_ number of apps in leading app stores 2021 _ Statista', Statista 2021, 2021. [Online]. Available: <https://www.statista.com/statistics/2766 23/number-of-apps- available-inleading-app-stores/>. [Accessed: 05-February-2021].

[4] Gu, Jie, et al. "Privacy concerns for mobile app download: An elaboration likelihood model perspective." Decision Support Systems 94 (2017): 19-28.

[5] Shabtai, Asaf, et al. "Google android: A comprehensive security assessment." IEEE Security & Privacy 8.2 (2010): 35-44..

[6] Shen, Yun, Pierre-Antoine Vervier, and Gianluca Stringhini. "Understanding worldwide private information collection on android." arXiv preprint arXiv:2102.12869 (2021)..

[7] Reardon, Joel, et al. "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system." 28th {USENIX} Security Symposium ({USENIX} Security 19). SANTA CLARA, CA, USA, (2019).

[8] Omar, Marwan, et al. "Android application security." Research Anthology on Securing Mobile Technologies and Applications. IGI Global, (2021), pp.610-625.

[9] Javed, Khadija, and Maria Tariq. "Formal modeling of security concerns in android." Lgurjcsit 4.1 (2020), pp.33-37..

[10] Marforio, Claudio, et al. "Analysis of the communication between colluding applications on modern smartphones." ACSAC '12: Annual Computer Security Applications Conference Orlando, Florida, USA,(2012), December 3 – 7.

[11] Memon, Atif M., and Ali Anwar. "Colluding apps: Tomorrow's mobile malware threat." IEEE Security & Privacy vol. 13, no.6, (2015): 77-81.

[12] Casolare, Rosangela, et al. "A model checking based proposal for mobile colluding attack detection." 2019 IEEE International Conference on Big Data (Big Data). IEEE, Los Angeles, CA, USA, (2019).

[13] Xu, Ke, Yingjiu Li, and Robert H. Deng. "Iccdetector: Icc-based malware detection on android." IEEE Transactions on Information Forensics and Security 11.6, (2016), pp1252-1264.

[14] Chin, Erika, et al. "Analyzing inter-application communication in Android." Proceedings of The 9th International Conference on Mobile Systems, Applications, and Services, Bethesda, Maryland, USA (2011),28 June - 1 July.

[15] Radzi, Wan Norsyafawati W. Muhamad, et al. "The impact of latest product features advanced technology on intention to purchase Android smartphones users." AIP Conference Proceedings, Vol. 2339. No. 1. (2021), AIP Publishing LLC, 2021.

[16] Alkindi, Zainab R., et al. "Android Application Permission Model." 4th Free & Open Source Software Conference (FOSSC'2019-OMAN),(2019).

[17] Negi, Charu, et al. "A Review and Case Study on Android Malware: Threat Model, Attacks,

Techniques and Tools." Journal of Cyber Security and Mobility (2021), pp231-260..

[18] Martinelli, F., Mercaldo, F., Nardone, V., Santone, A. and Vaglini, G., "Model checking and machine learning techniques for HummingBad mobile malware detection and mitigation", Simulation Modelling Practice and Theory, 105,(2020) p.102169,.

[19] Scoccia, Gian Luca, et al. "User-centric android flexible permissions." 39th International Conference on Software Engineering Buenos Aires Argentina, (2017, May 20 – 28.

[20] Andriotis, P., Li, S., Spyridopoulos, T. and Stringhini, G., "A comparative study of android users' privacy preferences under the runtime permission model" International Conference on Human Aspects of Information Security, Privacy, and Trust, Springer, Cham,(2017),pp. 604-622.

[21] Tiwari, A., Grob, S. and Hammer, C., "IIFA: modular inter-app intent information flow analysis of android applications", International Conference on Security and Privacy in Communication Systems, Springer, Cham, (2019), pp. 335-349,

[22] Sun, X., Li, L., Bissyandé, T.F., Klein, J., Octeau, D. and Grundy, J., "Taming Reflection: An Essential Step Toward Whole-program Analysis of Android Apps", ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 30, no.3 (2021), pp.1-36.

[23] You, W., Liang, B., Li, J., Shi, W. and Zhang, X., "Android implicit information flow demystified", 10th ACM Symposium on Information, Computer and Communications Security Singapore Republic of Singapore (2015), 14 April - 17 March, pp. 585-590.

[24] Chen, H., Tiu, A., Xu, Z. and Liu, Y., "A permission-dependent type system for secure information flow analysis", 2018 IEEE 31st Computer Security Foundations Symposium (CSF), Oxford, UK (2018), july 9-12, pp. 218-232.

[25] Mathis, B., Avdiienko, V., Soremekun, E.O., Böhme, M. and Zeller, A., "Detecting information flow by mutating input data" ACM/IEEE International Conference on Automated Software Engineering Urbana-Champaign IL, USA,(2017), 30 October- 3 November, pp. 263-273.

[26] Mercaldo, F., Nardone, V., Santone, A. and Visaggio, C.A., "Ransomware steals your phone. formal methods rescue it", International Conference on Formal Techniques for Distributed Objects, Components, and Systems,(2016), Springer, Cham, pp. 212-2

[27] Li, H., Shen, L., Wang, Y., Feng, J., Tan, H., & Li, Z. "Risk measurement method of collusion privilege escalation attacks for android apps based on feature weight and behavior determination", Security and Communication Networks, (2021).

[28] Casolare, R., Di Giacomo, U., Martinelli, F., Mercaldo, F., & Santone, A., "Android Collusion Detection by means of Audio Signal Analysis with Machine Learning techniques", Procedia Computer Science 192, (2021), pp. 2340-2346.