doi.org/10.54473/IJTRET.2023.7206

International Journal of Trendy Research in Engineering and Technology
Volume 7 Issue 2 April 2023
ISSN NO 2582-0958

ANDROID DATA LEAKAGE DETECTION SYSTEM

Abhinav Kumar & Shobhit Tyagi
Department of Computer Science, SET,
Sharda University Greater Noida, India
2019008151.abhinav@ug.sharda.ac.in , shobhit.tyagi@sharda.ac.in

Received 13 April 2023 Received in revised form 15 April 2023 Accepted 17 April 2023
Available online 25 April 2023

ABSTRACT

Android apps are either installed from the android market, from apk file, or are pre-installed in the ROM installed by the manufacturer. Private data from Android apps routinely escapes the device, whether intentionally or not. Numerous techniques have been proposed by researchers, such as static and dynamic analysis techniques to identify the apps most likely to leak personal information. In this study, we modify the dynamic analysis phase emulator using hooking technology. We are proposing the design, analysis, and implementation of an Android Data Leakage Detection System which detects the malware that's causing the data leak in android applications.

Keywords: *Data Leakage Detection System,Malware detection modeling methodology,MDMT,AndroZoo dataset*

I. INTRODUCTION

The number of smart gadgets is increasing exponentially in the modern world. These gadgets have different characteristics which offer users value-added service, these gadgets run typically on Android OS in order to use open-source software capabilities. Android is pretty famous in the community because it's open source and user-friendly and because of this popularity, android apps face several security issues. Android apps are compromised with different types of dangers that impact Android users in a negative way. The starting pace of malware apps per year is 78.6%. Around 8590 types of malware were found [1]. Strong analysis tools are the need of the hour to protect users from this android security issue. There are different ways to analyze android security issue, e.g. static analysis, which analyze Android app without execution and detects possible security risks [2].

To reduce the issue of android security breaches, several state-of-art approaches have been made to detect whether personal media have been transmitted or not, i.e. the is to see if any personal data has left the device [3]. What constitutes a privacy leakage by mobile apps, however, is a topic that has to be revisited in this age of mobile apps and cloud computing, many benign apps offer users services from the cloud. In order to send information to the cloud, these apps typically need to gather sensitive information like location and contacts. The same behavior, including sending sensitive data to the cloud, may be displayed by malicious programs that steal user data. [3].

Therefore, the communication of sensitive data alone may not indicate actual privacy leakage; a stronger signal should be whether or not the transmission is user-intentioned [3].

In general, Android applications can assume many behaviors; thus it is necessary to monitor their activities, for example, through interface or automatic event injectors [4]. We can also use hybrid analysis. A system that uses this approach must be created in such a way that, in the event that the first step fails, the second step would fill the void for the system to function properly. In mobile device, we use a standard approach to gather data optimally [4]. Android apps are either installed from the android market, from apk file, or are pre-installed in the ROM installed by the manufactured [5]. There are online shops that can examine Android applications during the publication process. Antivirus software installed on the device can also examine them, using a process or establishing a procedure. Before releasing an application, Google Play, for instance, employs Google Bouncer to check for any potential security flaws [5].

Recent approaches have employed one of three analytic methodologies, including static, dynamic, and hybrid analysis, to assess Android applications. Every analytical method can employ standalone application analysis. Static analysis is used to examine apps without actually running them [5]. Dynamic analysis methods require run programs on a virtual machine, emulator, or physical device, in contrast to static analysis methods. Applications allow us to get information that is not captured by static data analysis that takes into account user interface events,

dynamic content, human behavior, and device performance [5].

Recent studies also suggest that the developer should check the app on three cases of AppAudit First. AppAudit integrates with IDEs to check app for the developer before release [6]. Data leaks are typically caused by defective third-party modules, which may be identified in this way. Second, AppAudit is a service that can be implemented in app marketplaces to automatically audit apps. Due to AppAudit's excellent accuracy, market operators may completely automate the app auditing process by eliminating the need for humans to validate analytical results [6]. Due to AppAudit's high efficiency, developers no longer have to wait as long to receive auditing responses from the market after uploading their programs [6]. Third, before installing any apps, mobile devices can be equipped with AppAudit [6].

We are proposing the design, analysis, and implementation of an Android Data Leakage Detection System which detects the malware that's causing the data leak in android applications.

The rest of the paper has structured as follows. Section II reviews the existing literature. Section III explains the working of system analysis UML, Use case, and proposed algorithms. Section IV consists of system implementation algorithms and results. Section V consists conclusion and future work.

II. LITERATURE REVIEW

Casati et al examined the security vulnerabilities with several applications that are deemed important for handling user-sensitive data, demonstrating how an attacker may get a user access token that is kept on the device, potentially exposing users to identity theft. Authors cautioned that a man-in-the-middle (MITM) attack can be used by malevolent individuals to steal such a token as well as various other sensitive data. [1].

Shrivastava, & Kumar suggested approach which obtains permission clusters to quarantine the malware application using the k-means algorithm. For harmful behavior, an efficiency of 90% (about) is obtained, which validates the research. The usage of application permissions for possible applications in Android malware detection is supported by this study [2]. It is quite difficult to distinguish between malware and legitimate software because they both need the same kinds of permissions. By examining the permission patterns, a novel approach is suggested for detecting malware-based programs.

In the paper titled 'Android's sensitive data leakage detection based on API monitoring' [3] the authors approach the study of sensitive data by the analysis of sensitive APIs, decompiles Android APK to get small files, it establishes a delicate API library for user privacy. Then, analyzes the capacity threats via way of means of detecting the touchy API inside the supply code, and determines whether there may be a touchy information leakage

Cam, et al [5] in their paper ' Sensitive data leakage detection in pre-installed applications of custom android' report that transmission of touchy records does not really mean leakage of privacy, a higher indicator of this can we whether or not the transmission is through consumer-aim or not. The author has introduced a new framework for analysis which is called AppIntent. AppIntent can effectively deliver a series of GUI manipulations for each data transmission that corresponds to the series of occurrences that result in the data transfer, assisting a researcher in determining whether the data transmission was user-intentioned or not.

Cui et al [6] in their paper introduced a tool called CoChecker. Which states that employing static taint analysis, find the leak paths (chains of components) that would result in privilege escalation attacks. The author also proposes to construct a call graph to simulate the execution of numerous entry points in a component and get rid of false negatives brought on by the event-driven development style used by Android.

Chen et al [7] in their paper ' Automatic privacy leakage detection for massive android apps via a novel hybrid approach' have reported a novel hybrid methodology that, compared to existing static or dynamic algorithms, can identify more private data leakages. The strategy was realized in a program called HybriDroid,. Models for each app are extracted using both static and dynamic analysis techniques. The behavior model is then improved to reflect the results of the dynamic study. Cam, et al[8] have introduced a new system -uitHyDroid system. It enables the detection of sensitive data leakage via many applications using hybrid analysis. uitHyDroid. Static analysis is used by uitHyDroid to gather components of the user interface that must cooperate and shed light on any potentially sensitive data movements.

Xia et al[9] in their paper ‘ Effective real-time android application auditing’ have reported the designs AppAudit to overcome the problem of private data leakage. AppAudit's real-time app auditing is based on the integration of static and dynamic analysis. AppAudit represents a brand-new dynamic analysis technique that may replicate the execution of a portion of the program and carry out specific checks at each program state.

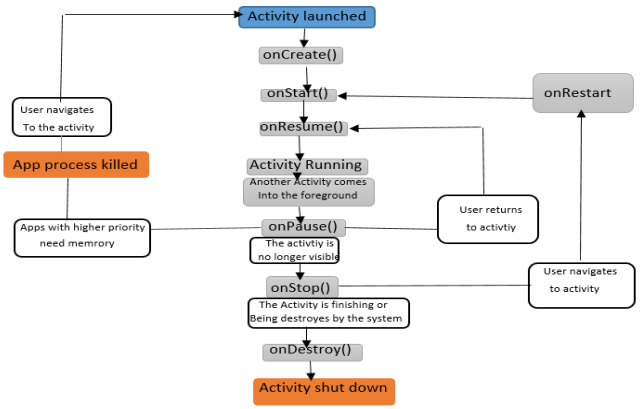
Then, this is used by AppAudit to reduce false positives from an effective but overly optimistic static analysis. Overall, AppAudit helps the app market by making app auditing useful in order to properly and efficiently identify data leaks, carriers, app developers, and mobile end users.

Kul et al [10] have stated that data leaks can be found via dynamic and static analysis. Using a dataset of Android applications and a data leak map. This strategy includes built-in permissions, activities, library classes, and methods. During dynamic analysis, the solution leverages logcat data and runtime permissions to identify the type of data leak and determine the likelihood that it would occur using an approach akin to k-nearest neighbors. This method further groups the leaks according to the degree of danger connected to that application.

III. SYSTEM ANALYSIS

Based on the aforementioned research, a thorough empirical investigation is offered. The system was created with consideration for the needs of the industrial partner. In this part, the system design is discussed.

In system analysis, we will cover the analysis of modules using the different Diagrams



.

Fig 1. Android Life Cycle

Fig 1. There are six main callback stages or lifecycle stages for an Android activity. OnCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy() are some of them. Each of these callbacks is activated by the system when an activity changes state.

All Android applications must implement the onCreate() callback. When we launch an activity from the home screen or intent, it is the first method called.

The onCreate() callback must be implemented in every Android application. It is the initial method called when we start an activity from the home screen or an intent.

OnResume() is immediately called after onStart(). This activity's components are all brought to the foreground state.

When a user goes to another activity or a multi-window mode program, onPause() is invoked.

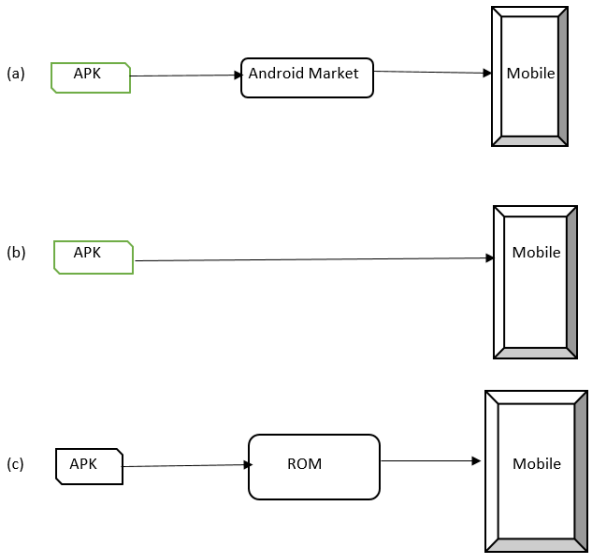


Fig 2. Three ways are used to put an application into a mobile device

Android apps can be installed in 3 ways (1) through Google PlayStore (2) Trough APK files and (3) Through the pre-installed applications installed in the original ROM. The above Fig 2. is the third part, these are the pre-installed applications. They are typically impossible for the end user to uninstall because they are installed in the

International Journal of Trendy Research in Engineering and Technology
Volume 7 Issue 2 April 2023
ISSN NO 2582-0958

ROMs during the ROM compilation process. They are ROMs provided in two ways to users. Before distribution, they are first installed on mobile devices. Users can also download ROMs from the Internet as a second option.

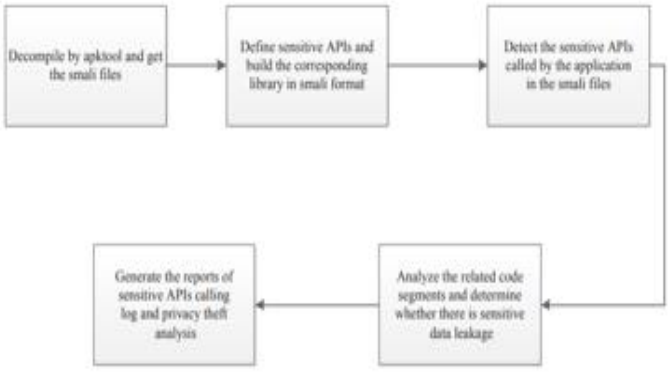


Fig 3. Flow of sensitive API calls direction and malicious behavior of analysis

Fig 3. shows the flow of sensitive API calls for camera detection and malicious behavior analysis, GPS location, Bluetooth activation, and other functions are classified as delicate APIs.

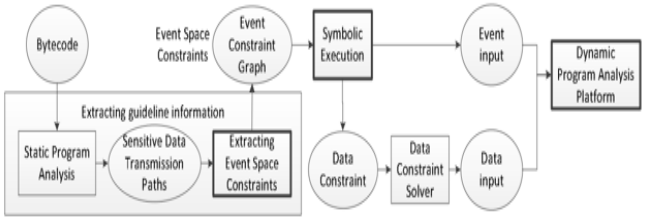


Fig 4. The overall architecture of AppIntent

The above figure Fig 4. describes the overall architecture of AppIntent, which analysis the target app in 2 steps:

A.Guided Symbolic Execution using Event–Space

The first stage is to create crucial inputs that require the delivery of sensitive data. To preprocess and extract all potential data transmission paths as well as potential events connected to each, we use a static taint analysis path, which aids in building a graph of event-space constraints. The guided symbolic execution then uses the graph to extract crucial inputs. Meanwhile, the nature of the symbolic method ensures code coverage.

B.Dynamic Program Analysis Platform

Although they accurately state the circumstances in which transmission would occur, the inputs generated in the first phase are not comprehensible enough. These inputs help us adapt Android. In order to better understand sensitive data propagation, InstrumentationTestRunner is used to automate the app's step-by-step execution. This mimics user interactions with UI changes. We think it can successfully depict the source of the communication so that we can determine whether or not the broadcast was intentionally made by the user.

IV. SYSTEM IMPLEMENTATION AND RESULT

In this part, we will be studying the algorithms and methods that we will be using to implement the Android Data Leakage Detection System, which, in turn, will enable us to comprehend the overall architecture of Data Leakage Detection in Android.

Android is free software that stores the private information of its users. It is challenging to restrict Android usage which is dishonest due to the breadth of its application scope. Attackers are currently trying to penetrate the Android Market using malware that takes data and wrongly impersonates the user. Although some research is being done in this area, many other areas have not yet been investigated.

There is a method for detecting malware that uses data analytics methods. This method is known as MDMT, or malware detection modeling methodology, and it takes Android applications' permissions into account. Now that malware is focusing on the internet, current security risks face more difficulties. As a result, combating malware becomes challenging, and the best methods for its detection and prevention are urgently required.

A.Collecting Sample Data

Massive amounts of data are being downloaded, and the procedure takes a long time. The above-mentioned issue can be solved by using "AndroZoo," an online resource that delivers the necessary information when you need it. The AndroZoo dataset (Allix, Bissyandé, and others) is used to analyze the suggested approach. The database (Klein, & Le Traon, 2016) has more than 50 lahks Android applications. The whole dataset collected from AndroZoo is used by VirusTotal (n.d.) to identify genuine malware

apps because AndroZoo does not release the labelled dataset. After being obtained from AndroZoo, the application is eventually uploaded to VirusTotal for the purpose of determining the vt detection value. 1038 apps are used from AndroZoo for the analysis. These applications are labeled as malware based on the vt detection value provided by AndroZoo.

```
#Algorithm 1 : Data Extraction

1. read SHA256 file.

2. for (i = 1 to 1038)  xf  ← SHA256_values test ← strcat (xf, API_key) url ← vector (test)

3. for(i = 1 to length (xf)) get (url(i))//download.apk file using get method.
```

Fig 5. Algorithm 1 (Extraction of Data)

Clusters of malware-free and clean Android applications are discovered after pre-processing the acquired data.

B.Reverse Engineering

Every Android application has an AndroidManifest.xml file, which includes metadata, services, features, and permission names. Reverse engineering is used to obtain this XML file, as demonstrated in Figure 3. This process turns the XML file that was extracted from the APK file into readable form.

C.Extraction of Permission

The permission that the application needs is one of the many features and pieces of metadata found in the AndroidManifest.xml file. These permissions are used by the suggested algorithm to foretell an application's malevolent intentions. By removing the word "uses permission" from an XML file, one can extract permissions from the AndroidManifest.xml. In order to acquire the trimmed permission name, the artifacts are also eliminated.

D.Permission-Based Clustering

Three distinct permissions clusters are created using this research based on high, low, and similar frequency differences. Then, as shown in Algorithm 2, the k-means method is used to further classify a comparable frequency cluster into two distinct clusters.

```
#Algorithm 2 : Permission frequency

1. Initialize i, j, k, and l.

2. n_clus ← read the file of n pair cluster

3. bool ← empty vector

4. boo[] ← empty list
```

```
5. for (I = 1 to I <= n_clus)
     for (j = 1 to j <= all_perm)
        c ← 0
        d ← 0
        new ← artifact (permission)
        for (k = 1 to k <= new)
         now iterate perm1 till length of new
          if match exists
           c++
          now iterate perm2 till length new
           if match exists
             d++
          if (c+d == 2)
           bool = 1
          else
           bool = 0
        boo[l] = bool
        l++
        bool = 0
```

Fig 6. Algorithm 2 (Permission Frequency)

A. Permission Combinations

Malware has a variety of behaviors. Therefore, the vectors of various permission combinations are created and evaluated on experimental data in order to track the behavior of the malware.

We have run all these algorithms on our system. These algorithms result in providing the comfort running of a system.

International Journal of Trendy Research in Engineering and Technology
Volume 7 Issue 2 April 2023
ISSN NO 2582-0958

Algorithm 1 and Algorithm together combine to form our Android Data Leakage Detection System.

The Machine in which we implemented the system contained 8 GB RAM, 4 GB GeForce GTX Graphics, I5 Processor, and 1 TB SDD and the software we used was Visual Studio and the languages which have been used to implement the system were HTML, CSS, Python, and Database used was MongoDB i.e. Mongoose and cloud database is MongoDB Atlas.

Our Project is divided into 2 parts – frontend and backend. The front end most part is done using React JS and some is done using HTML, CSS, and Bootstrap while the backend part is done using Python and the database is MongoDB.

All these algorithms are implemented using Python on the server side of the code. Thus, the result obtained from the algorithms helped us in making the platform for web-based Android Data Leakage Detection System. All of the Algorithms have improved its performance to the utmost level.

V. CONCLUSION

This study suggested a method for analyzing Android apps to determine if they are clean or malicious. The suggested method examines the malware application's mix of rights and quarantines. Here, k = 2, 3, 4, and 5 permission combinations are used, and these combinations are compared to malware behavior. A success percentage of 89.5% is proven by successfully detecting the malware behavior while analyzing the real malware dataset. The accuracy of 22.5% for the identical vector set when tested on a clean dataset proved that the vector used was similar to a harmful pattern.

The accuracy of 22.5% for the identical vector set when tested on a clean dataset proved that the vector used was similar to a harmful pattern. In order to improve success rates, we will examine Android program permissions in conjunction with intent in the future. The suggested approach for handling massive data can be combined with Z-tests and other Android capabilities, and it may eventually be expanded to other operating systems.

REFERENCES

[1] Casati, L., & Visconti, A. ‘ The dangers of rooting: data leakage detection in Android applications’, Mobile Information Systems, Special Issue(2018)1-9.

[2] Shrivastava, G., & Kumar, P.’Android application behavioural analysis for data leakage. Expert Systems, 38(2019), e12468.

[3] Shanshan, M., Xiaohui, Y., Yubo, S., Kelong, Z.,Fei, C.,’. Android's sensitive data leakage detection based on API monitoring’, Conference: Proceedings of the 2013 Fifth International Conference on Multimedia Information Networking and Security.

[4] Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P., & Wang, X. S. (2013, November). Appintent: Analyzing sensitive data transmission in android for privacy leakage detection’, Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (2013) 1043-1054

[5] Cam, N. T., Pham, V. H., & Nguyen, T.,’Sensitive data leakage detection in pre-installed applications of custom android firmware’,. 18th IEEE International Conference on Mobile Data Management (MDM)(2017) 340-343 IEEE.

[6] Cui, X., Yu, D., Chan, P., Hui, L. C., Yiu, S. M., & Qing, S. ‘Cochecker: Detecting capability and sensitive data leaks from component chains in android’, Australasian Conference on Information Security and Privacy (2014). 446-453.

[7] Chen, H., Leung, H. F., Han, B., & Su, J. ‘Automatic privacy leakage detection for massive android apps via a novel hybrid approach’, IEEE International Conference on Communications (ICC)(2017) 1-7.

[8] Cam, N. T., Pham, V. H., & Nguyen, T. ‘Detecting sensitive data leakage via inter-applications on Android using a hybrid analysis technique’, Cluster Computing 22(2019), 1055-1064.

[9] Xia, M., Gong, L., Lyu, Y., Qi, Z., & Liu, X, ‘Effective real-time android application auditing’, IEEE Symposium on Security and Privacy (2015)899-914.

[10] Kul, G., Upadhyaya, S., & Chandola, V, ‘Detecting data leakage from databases on android apps with concept drift’,17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (2018) 905-913