Whether Android Applications Broadcast Your Private information: A Naive Bayesian-based Analysis Approach

Li Lin[1,2,3] , Jian NI[1,2,3], Xinya Mao[1,2,3],Jianbiao Zhang[1,2,3]
[1]College of Computer Science, Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China
[2]Beijing Key Laboratory of Trusted Computing, Beijing 100124, China
[3]National Engineering Laboratory for Classified Information Security Protection, Beijing 100124, China
linli_2009@bjut.edu.cn, {17801028290, 13089397966}@163.com, zjb@bjut.edu.cn

***Abstract*—With the rapid development of android smart terminals, android applications are exhibiting explosive growth. However, there remains a challenging issue facing android system, a malicious application may broadcast user's private information. In this paper, we propose a Naive Bayesian-based approach for analyzing private information leakage under the android broadcast mechanism, which calls BRbysA. Firstly, broadcast actions registered in manifest.xml are picked up statically by keyword matching technique. Secondly, with the Xposed framework, the broadcast actions specified at run time are discovered by hooking broadcast callback onReceive() function. Combining the above two ways, we can capture all real-time broadcast actions in an android application. Thirdly, we adopt Naive Bayesian learning algorithm, all broadcast actions which involved in users' privacy leakage are analyzed and classified. Finally, we evaluate the proposed approach by using the dataset from Drebin and Google Play.**

Keywords- Android broadcast; private information leakage; keyword matching; hook; Naive Bayesian

I. INTRODUCTION

Nowadays more attention has been paid to individual privacy information protection in android platform [1]. There are different security capabilities in android custom ROM [2]. It leads to the potential loopholes and vulnerabilities in android system [3]. For example, a loophole in BlackPhone has been discovered to allow an attacker to decrypt users' encrypted information [4]. There has occurred a new malware through Google Play Store to inject malicious codes for collecting the user's bank and credit card information [5]. Consequently, it is of great importance to underpin trusted android market by malicious applications analysis approach, which can accurately and efficiently detect APKs that reveal users' private information.

Existing methods are mainly divided into two categories: static detection and dynamic monitoring. Static detection methods involve in the analysis of signature information and permission information. However it is difficult to find the privacy disclosure from android broadcast mechanism by analyzing permission information. Dynamic detection methods like MonkeyRunner and HIPS can achieve malicious behavior detection with higher detection accuracy, but it also suffer from low code coverage and longtime consumption. In fact, android system may broadcast users' private information [6]. If a malicious program can accept system broadcast, it can obtain users' private information by calling onReceive() method.

To solve these problems, this paper proposes a Naive Bayesian-based approach called BRbysA for analyzing private information leakage under the android broadcast mechanism. Firstly, broadcast actions registered in the manifest.xml are picked up statically by exploiting keyword matching technique. Secondly, referencing Xposed framework, the broadcast actions specified at running time are discovered by hooking broadcast callback onReceive() function. Combining the above two ways, we can capture all real-time broadcast actions in an android application. Thirdly, adopting Naive Bayesian-based learning algorithm, all broadcast actions involved in users' privacy leakage are analyzed and classified. To our knowledge, we are the first one to introduce the machine learning-based method to analyze android broadcast receiver. Furthermore, based on the above analysis, the probability of each broadcast actions appeared in malicious programs is calculated to determine whether the application discloses users' private information Based on Drebin dataset proposed by Daniel et al [7] and Google Play application set, we have test the effectiveness of the proposed method successfully. We summed up several kinds of broadcast actions that really exist the risk of leaking users' private information.

The rest of the paper is organized as follows. Section II presents related work. Section III gives the problem description. Section IV introduces the proposed method in detail. Section V presents our experimental results. Section VI concludes this paper and shows some possible future work.

II. RELATED WORK

In the aspect of android malware detection, there are two schemes: static detection and dynamic monitoring. Static detection methods [8] mainly concern the analysis of signature information, permission information of APKs. Dynamic

DOI reference number: 10.18293/SEKE2018-208

detection methods [9], [10], [11] can trigger the malicious behavior of malicious programs by running android applications. Common dynamic testing tools include Monkey, MonkeyRunner, TaintDroid, droidBox and so on [12].

Most of current malicious applications detection researches focus on the analysis of permissions and malicious APIs. There was a little work on investigating android broadcast mechanism. Fadi Mohsen et al. [6] have developed a tool broadcastViewer, which can be used to view android system broadcast actions that has been installed on your phone. However, the work detects broadcast actions only from the perspective of static analysis and cannot detect the behavior of broadcast receiver when the actual operations are triggered by the application. In addition, Erika Chin et al. [13] proposed ComDroid to analyze android inter-process communication risk. Di Tian et al. [14] analyzed the characteristics of Broadcast Receiver vulnerability and developed a broadcast receiver vulnerability detection system called BRVD. Noam Kogan et al. [15] present an anti-pirate revocation scheme for broadcast encryption systems. The above three work have mentioned that broadcast message may be with user's private information in broadcast transmission and inter-application communication. But they also didn't analyze all broadcast actions systematically. It may lead to miss some broadcast actions that reveal users' private information.

III. PROBLEM DESCRIPTION

Users download a wide variety of APKs from third-party application market in order to meet their daily commerce and communication needs [16]. An android system consists of four components: BroadcastReceiver, Activity, Service, and Content provider. BroadcastReceiver is used to receive broadcast declared in Manifest.xml file. At this time, BroadcastReceiver also can call its onReceive() method to perform certain operations. Interaction between BroadcastReceiver and android system is shown in Fig. 1. The broadcast information may contain users' private information. A malicious program can perform android broadcast callback function by customizing onReceive() method. For example, if a malicious program has registered SMS BroadcastReceiver, it can receive SMS messages and access the information through intent.getExtras() method. In this paper, our objective is to provide a sound method for detecting and analyzing malicious android APKs, which use broadcast to steal user privacy.

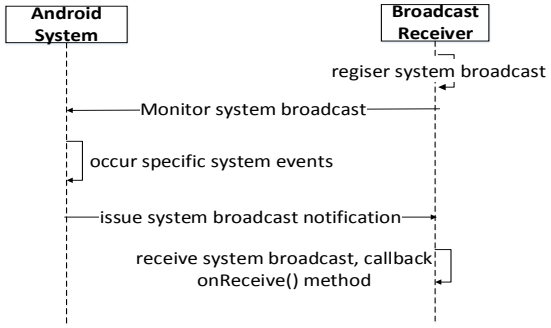

Figure 1. Interaction between broadcast receiver and android system

IV. DESIGN OF BRBYSA

As shown in Fig.2, the BRbysA method is divided into three parts: broadcast action static collection, broadcast action dynamic discover and Naive Bayesian-based privacy leakage analysis. It must combine static and dynamic methods to collect broadcast action, because the existing dynamic technologies do not monitor all broadcast actions in the APK's manifest.xml. Moreover, exploiting Naive Bayesian-based learning algorithm, we can determine whether there exist the risk of leaking users' privacy through broadcast action information in an APK.

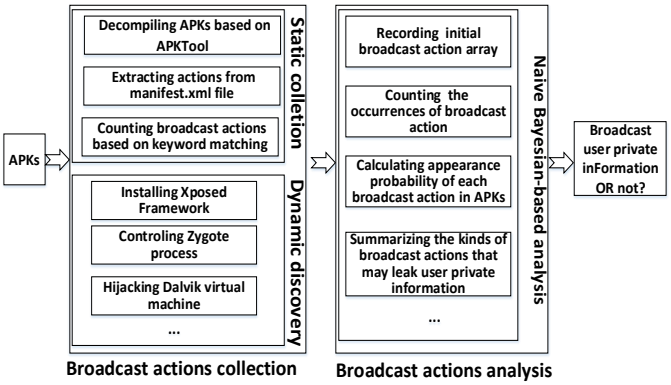

Figure 2.The principle of BRbysA

A. Statically collecting broadcast action

In general, self-defined BroadcastReceiver is specified by receiver label. A particular broadcast action can be specified by intent-filter label and received by BroadcastReceiver. In BRbysA, broadcast actions in APK's manifest.xml file are collected using keyword matching algorithm in Table I.

TABLE I. KEYWORD MATCHING-BASED BROADCAST ACTION EXTRACTION ALGORITHM

Input: APK package under test
Output: broadcast action set

Begin:
1. Decompile APK source file based on APKTool.
2. Extract the manifest.xml file of APK into a specific directory.
3. Read the manifest.xml using FileReader and BufferdReader.
4. Split the manifest.xml across blank, store elements into String array.
5. Create broadcast actions set, to compare with String array

```
while ( bufferdReader.readLine() != null) {
    String[] words = line.split(" ");
    for (String word : words) {
    if (word.equals("SMS_RECEIVED"))
    {arrayList.add("SMS_RECEIVED")}
    }
}
```

6. If the broadcast action equals the array element, then add it to the result set.

End.

B. Dynamically discovering broadcast action

To obtain broadcast actions specified at run time, this paper proposes a hook-based dynamic monitoring scheme to observe the actual state of broadcast callback onReceive() method. The hook operation for broadcast registration API is carried out based on Xposed framework. BRbysA states the XposedInstaller entrance class in the configuration file assets/xposed_init. The algorithm is shown in Table II.

TABLE II. BROADCAST ACTION DYNAMICAL DISCOVERY ALGORITHM BASED ON XPOSED FRAMEWORK

Input: APK package under test
Output: broadcast action set

Begin:
1. Android cellphone install Xposed framework, then configure Xposed framework in project's manifest.xml.
2. Import XposedBridgeApi-54.jar into project as library file, the stating XposedInstaller entry class in the assets/xposed.init file.
3. Main class Module implements IXposedHookLoadPackage Interface, override handleLoadpackage(LoadPackageParam lpparam) method.
4. Assign the hooked package name、method name and param type, across XposedHelpers.findAndHookMethod().
5. When the specific package is loaded, the system will call findAndHookMethod(), and then call beforeHookedMethod and afterHookedMethod.
6. Get broadcast action through ((Intent)param.arg[1]).getAction to, then storage it into broadcast action array.

End.

C. Naive Bayesian-based privacy leakage analysis

Based on the above methods, all real-time broadcast actions can be captured in an APK. Next, a naive Bayesian machine learning algorithm is introduced to classify broadcast actions. Let X_0 be normal program, X_1 be malicious program, C_i represents broadcast action ba_i appearing in an APK, then the formula $P(C_i|X_i) = P(X_i|C_i) * P(C_i)/P(X_i)$ (1) indicates the occurrence probability of broadcast action ba_i when the APK is normal or malicious. The algorithm is shown in Table Ⅲ.

TABLE III. NAIVE BAYESIAN-BASED PRIVACY LEAKAGE ANALYSIS ALGORITHM

Input: Broadcast action set outputted by the above methods
Output: the conditional probability of occurrence of each broadcast action when an APK is malicious

Begin:
1. Construct an array ActionArrays[], each element of ActionArrays[] is from the broadcast action set outputted by the above methods.
2. Array ClassVec[] stores APKs' classification, where '0' indicates normal program, '1' indicates malicious program. The array ArraySingle is used to store the occurrence number of unreplicated broadcast action . The initial elements of ArraySingle are set all '0'.
3. For any broadcast action ba_i, count the number of malicious APKs from ActionArrays under the premise of ba_i occurrence, which is denoted by N_{i1}, count the total number of all APKs from ArraySingle under the premise of ba_i occurrence, which is denoted by N_{i2}. Calculate $P(X_1|C_i)$ through N_{i1} divided by N_{i2}.
4. For any APK, calculate $P(X_1)$, the probability that APK is malicious.
5. For any broadcast action ba_i, count the number of ba_i in ArraySingle N_{i3}; Count the total number of all broadcast action in ActionArrays N_{i4}. Calculate $P(C_i)$ through N_{i3} divided by N_{i4}.
6. For any broadcast action ba_i, calculate $P(C_i|X_1)$ by Formula (1).
7. For different broadcast actions in an APK, count the average value of $P(C_i|X_1)$ and find a suitable threshold. Determine whether an APK will reveal user private information by judging if the above average value exceeds the selected threshold or not.

End.

V. EXPERIMENT EVALUATION

A. Experiment environment

We have implemented and deployed *BRbysA* on Android version 4.4. The total kinds of broadcast actions is 136. The Xposed Framework version used in *BRbysA* is No.54. We randomly choose 1000 regular APKs from the Google Play and download 1000 malicious APKs from Drebin dataset as experimental samples.

B. Static collection and dynamic discovery effectiveness

Using the algorithms in Table I and II, we have counted the number of all broadcast actions in the 2000 APKs. Due to limited space, here we only release statistic data. As shown in Table IV, only 31 in 136 kinds of broadcast actions appears in the 2000 APKs, the others don't appear. PA represents the average value of conditional probability for broadcast actions occurring under the premise of malicious APK. The result shows that the first five have higher frequency than 0.3.

TABLE IV. THE NUMBER OF ALL BROADCAST ACTIONS IN THE SAMPLE.

Broadcast action	Num.	PA
android.provider.Telephony.SMS_RECEIVED	300	0.4507
android.net.conn.CONNECTIVITY_CHANGE	243	0.4105
android.net.wifi.NETWORK_IDS_CHANGED	156	0.3954
android.intent.action.PHONE_STATE	113	0.3522
android.intent.action.NEW_OUTGOING_CALL	84	0.3013
Intent.ACTION_LOCALE_CHANGED	53	0.2652
Intent.ACTION_REBOOT	22	0.1024
android.hardware.action.NEW_PICTURE	7	0.0202
android.hardware.action.NEW_VIDEO	4	0.0106
android.intent.action.CAMERA_BUTTON	2	0.0103
android.intent.action.DATA_SMS_RECEIVED	2	0.0096
android.intent.action.FETCH_VOICEMAIL	2	0.0043
android.intent.action.NEW_VOICEMAIL	1	0.0043
android.intent.action.PROVIDER_CHANGED	1	0
android.intent.action.PROXY_CHANGE	1	0
android.intent.action.SCREEN_ON	1	0
android.intent.action.SCREEN_OFF	1	0.0043
android.media.VIBRATE_SETTING_CHANGED	1	0.0043
android.net.nsd.STATE_CHANGED	1	0
android.net.wifi.STATE_CHANGE	1	0
android.nfc.action.ADAPTER_STATE_CHANGED	1	0
android.provider.Telephony.SMS_RECEIVED	1	0.0043
android.intent.action.WALLPAPER_CHANGED	1	0
android.media.AUDIO_BECOMING_NOISY	1	0
android.intent.action.PACKAGE_FIRST_LAUNCH	1	0.0043
android.intent.action.MEDIA_SCANNER_SCAN_FIL	1	0.0043
android.intent.action.INPUT_METHOD_CHANGED	1	0
android.intent.action.BATTERY_LOW	1	0
android.intent.action.BATTERY_CHANGED	1	0
android.bluetooth.device.action.UUID	1	0.0043
android.bluetooth.devicepicker.action.LAUNCH	1	0.0043

C. Malicious APKs detection accuracy

Next, we use the above samples to test malicious APK detection accuracy of *BRbysA*. As shown in Fig. 3, the horizontal axis represents probability value ranges, the vertical axis represents the number of APKs that PA falls in a certain probability value range. In the experiment, 0.3 is selected as a threshold. As shown in Fig. 3, the number of APKs whose PA is greater than 0.3 is 123+141 + 26 = 290. That means, 290 malicious APKs can be detected by *BRbysA*. Meanwhile, the number of APKs registered broadcast actions in 1000 malicious APKs is counted as 310. Considering the most extreme situation, all malicious APKs registered broadcast actions exist the risk of leaking users' private information. Thus the malicious APKs detection rate of *BRbysA* is 290/310=93.548%. The result shows that *BRbysA* can achieve good malicious APKs detection.

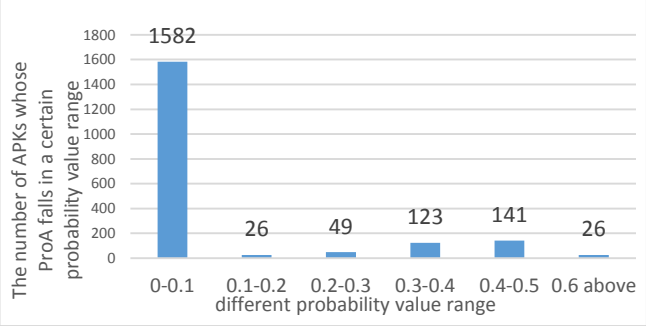

Figure 3.Different probability value range vs the number of APKs whose ProA falls in a corresponding probability value range.

Finally, we compare *BRbysA* with Manilyzer [17]. In the experiment, we randomly select 100 APKs from the above 310 malicious APKs registered broadcast actions and select 100 APKs from the above 1000 regular APKs. *BRbysA* and Manilyzer are implemented to analyze the selected 200 APKs. As shown in Fig. 4, the horizontal axis represents different rounds and the vertical axis represents the number of detected malicious APKs under different malicious APKs detection methods. The result shows that *BRbysA* can work better than Manilyzer at malware detection rate in most cases.

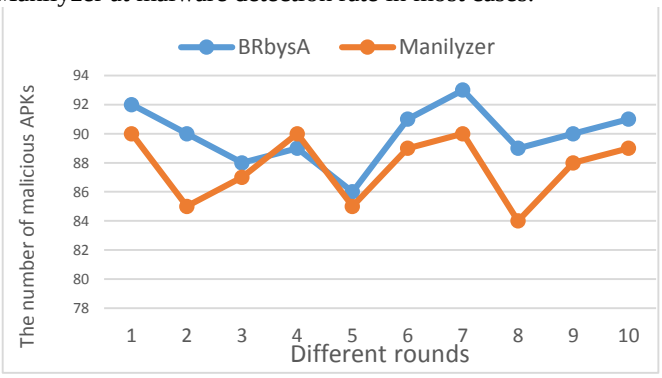

Figure 4. Different rounds vs the number of detected malicious APKs under different malicious APKs detection methods.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposes *BRbysA* to analyze private information leakage under android broadcast mechanism. The highlights of this paper are as follows. Firstly, combining static collection and dynamic discovery, all real-time broadcast actions in an APK can be captured. Secondly, adopting Naive Bayesian learning algorithm, all broadcast actions involved in users' privacy leakage can be analyzed and classified. Finally, we have evaluated *BRbysA* using the dataset from Drebin and Google Play, which illustrate there are several kinds of broadcast actions that really exist the risk of leaking users' private information.

However, the problem of threshold selection is not covered in this paper. With the change of sample size, we have found that different thresholds may make different decisions. This is desirable to develop some data mining approaches to increase the malware detection accuracy in the set of more malwares. Furthermore, our ongoing research will test the time and load overhead performance of the proposed approach.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation of China (No. 61502017), the Scientific Research Common Program of Beijing Municipal Commission of Education (KM201710005024).

REFERENCES

[1] Zhang. W, Li.X, Xiong.N and Vasilakos. A. V, "Android platform-based individual privacy information protection system," *Personal and Ubiquitous Computing*, Vol. 20, no. 6, pp. 875-884, 2016.

[2] Xu. M, Song. C,Ji. Y, Shih. M, Lu. K, Zheng C, et al, "Toward Engineering a Secure Android Ecosystem: A Survey of Existing Techniques," *ACM Computing Surveys*, Vol. 49, no. 2, pp.38, 2016.

[3] Yang. K, Zhuge.J, Wang. Y, Zhou. L, and Duan. H, "IntentFuzzer: detecting capability leaks of android applications," *in Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp.531-536, ACM, 2014.

[4] 360 security broadcast, The world's most secure mobile phone BlackPhone was found loopholes can cause loss of privacy [Online]. Available: http://bobao.360.cn/news/detail/1171.html 2015-01.

[5] Ren. C, Chen. K, and Liu. P, "Droidmarking: resilient software watermarking for impeding android application repackaging," *in Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pp.635-646, ACM, 2014.

[6] F. Mohsen, M. Shehab, E. Bello-Ogunu,and AA. Jarrah, "Android System Broadcast Actions Broadcasts Your Privacy," *in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp.1484-1486, ACM, 2014.

[7] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon and K. Rieck , "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket,"*in Proceedings of NDSS 2014*, Internet Society, 2014.

[8] Quan. D, Zhai. L, Yang.F and Wang. P, "Detection of Android Malicious Apps Based on the Sensitive Behavious," *in Proceedings of TrustCom 2014*, pp. 877-883, IEEE, 2014.

[9] S. S. Shinde and S. S. Sambare. "Enhancement on privacy permission management for Android apps," *Communication Technologies*, pp.838-842, IEEE, 2015.

[10] Yang.W, Xiao. X, P. Rahul, E. William and Xie.T, "Improving mobile application security via bridging user expectations and application behaviors," *in Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, pp.32, ACM, 2014.

[11] S. Feese, B. Arnrich, G. Troster, M. Burtscher, B. Meyer and K. Jonas, "CoenoFire: monitoring performance indicators of firefighters in real-world missions using smartphones," *in Proceedings of the ACM international joint conference on Pervasive and ubiquitous computing*, pp.83-92, ACM, 2013.

[12] S. K. Singh, B. Mishra and P. Gera, "A privacy enhanced security framework for android users,"*in Proceedings of the 5th International Conference on IT Convergence and Security*, pp.1-6, IEEE, 2015.

[13] E. Chin, A. P. Felt, K. Greenwood and D. Wagner, "Analyzing Inter-Application Communication in Android," *in Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services, MobiSys'11 and Co-located Workshops*, pp. 239-252, ACM, 2011.

[14] Tian.D, "Detecting Vulnerabilities of Broadcast Receivers in Android Applications," *ProQuest Dissertations and Theses A&I: The Sciences and Engineering Collection*, 2016.

[15] N. Kogan, Y. Shavitt and A. Wool, "A practical revocation scheme for broadcast encryption using smartcards," *ACM Transactions on Information and System Security (TISSEC)*,Vol.9,no.3,pp.225, 2006.

[16] Li. Y, Yao. F, Lan. T and Venkataramani. G. , "SARRE: Semantics-Aware Rule Recommendation and Enforcement for Event Paths on Android," *IEEE Transations on Information Forensics and Security*: Vol.11, no 12, pp. 2748-2762, 2016.

[17] S. Feldman, D. Stadther and B. Wang, "Manilyzer: Automated Android malware detection through manifest analysis," *in Proceedings of the 11th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 767-772, IEEE, 2015.