Journal of Theoretical and Applied Information Technology
20th September 2013. Vol. 55 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A SURVEY ON MITIGATING ATTACKS RELATED TO SHORTCOMINGS OF ANDROID PERMISSION FRAMEWORK

[1]IMAN KASHEFI, [2]MAZLEENA SALLEH

[1]Faculty of Computing (FC), Universiti Teknologi Malaysia (UTM), 81300 Johor Bahru, Malaysia
[2]Faculty of Computing (FC), Universiti Teknologi Malaysia (UTM), 81300 Johor Bahru, Malaysia
E-mail: [1]kaashefi@gmail.com , [2] mazleena@utm.my

ABSTRACT

Today Smartphones are the closest user assistants since they offer a wide range of functionalities to users. Although there are many applications developed in the market which facilitate the day-to-day user activities and provide a comprehensive means to entertain users, the number of malicious applications which misuse the users' personal data or overcharge them are increased accordingly. These applications are granted privileges legitimately while they may not use them in a proper way. The aim of this paper is to address attacks related to the shortcomings of Android permission framework, which further are categorize to attacks result from applications with excessive privileges, confused deputy, and collusion attacks. This work compares the ability of existing approaches in mitigating these attacks since any improvement in current mechanisms or proposing novel methods to impede these types of attacks would not be achieved unless a comprehensive study on the current approaches takes place.

Keywords: *Android Security, Privilege Escalation Attack, Collusion Attack, Confused Deputy Attack*

1. INTRODUCTION

Although Androids inherits many security countermeasures from Linux, and also it retrofits the framework with specific security mechanisms, researches [1, 2] shows that like any other computer device in network system, it is threaten by many different kinds of attacks. One of the most important attacks that jeopardize the users' privacy or over charge users comes from installing applications from different markets and also Google official market, "Google Play". Unfortunately recent researches [3, 4, 5, 6, 7] showed that currently there are various malicious applications uploaded in market which misuse the deficiencies in Security framework, and users are attracted by their splendid advertisement. These applications have been developed with malicious purpose such as leaking user sensitive information [8]. Applications are granted privileges legitimately and users are not aware what is doing in the background and whether their private information is using in a proper way or not. However the problem of privilege escalation attacks does not limit to over-privilege applications; attacks. Confused deputy attacks [9] discuss scenarios where a malicious application misuses the vulnerable interfaces of another benign privileged (but not well-protected) application. On the other hand, colluding attack [9] concerns with two or more applications which share their privileges in order to empower them to perform actions beyond their individual permissions. This paper with expressing the Android architecture and security mechanism aims to address the deficiencies in its frameworks which are exploited by attackers to perform privilege escalation attacks. Moreover it discusses many works, which are mainly in the form of extensions to android framework, aim to prevent the above kinds of attacks. The rest of this paper is organized as follow: In section II, a brief review of Android Architecture is presented, section III explained the Android security, then in section IV, some of the most famous works aim at extending the Android security mechanism will be discussed, finally section V concludes this paper and provide some open area for future research.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. ANDROID ARCHITECTURE

Android is a Linux-based, open source, mobile phone platform that includes an Operating System (OS), middleware, and key applications [10] as it is demonstrated in Figure 1.

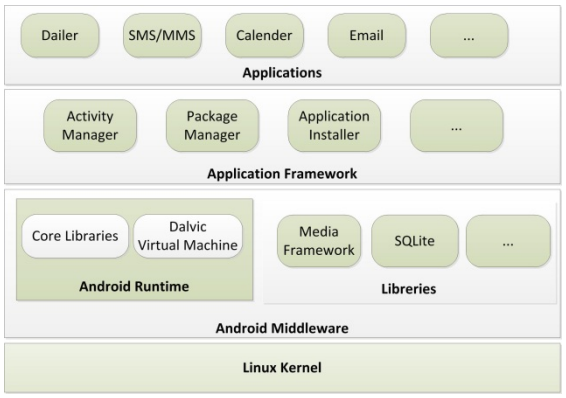

Figure 1: Android Software Stack

The foundation of the Android software stack is the *Linux Kernel*. Android uses Linux for its device drivers, memory management, process management, and networking. In the conceptual model, the kernel layer is placed between hardware and the software layers to provide core functionalities for Android services [10]. The middleware itself is written in Java and C/C++. The application framework includes applications written in C/C++ or Java that provide functionalities for system purpose.

Android Runtime, in the *Android Middleware* layer, consists of the Dalvik Virtual Machine and the core libraries. Dalvik runs .dex (Dalvik Executable) files which are more compact and memory-efficient than Java class files.

The *Application Framework* layer encompasses Google-supplied tools along with proprietary extensions or services. One of the main components of the framework is the Activity Manager, which controls the lifecycle of applications. The Package Manager is responsible for assigning privileges to applications at install time. The Package Manager also verifies the accuracy and completeness of the .apk files. Package installer extracts and installs .apk files according to the Android installation mechanism. Applications in Android are packaged in an .apk (Android package) archive. The .apk is alike to a standard Java jar file that it holds all code and non-code resources such as images, sound, manifest, and so forth. The top layer is the *Applications* layer for implementing applications such as email client, calendar, phone and so forth.

The Android package is composed of different components. Components in a package can access to the components of other packages and share data only through the ways provided by the system. Every Android package is associated with a primary process in which components of an application such as activity, service, content provider, and broadcast receiver are executed. Activities present the user interfaces (or screens) of an application, generally each screen which is presented to a user is proposed only by a single Activity. Services control backbone processing and in contrast to activity components, they are hidden to the user. Content Provider is in charge for store and share data. Each content provider comes with a relevant authority which describes the regulations on the component from other application which has permissions to read or write data associated with it. Broadcast Receiver is constructed in the form of mailboxes to receive messages from other applications. Applications are enabled to broadcast messages to an implicit or explicit destination. In implicit broadcasting an application can only receive messages from the destinations for which has subscribed [2, 11].

The communication between components occurs through an inter component communication (ICC) mechanism intent messages are used in this mechanism. An intent is a data structure which contains the information about the intended action needs to be performed. There are two types of intents; explicit intents and implicit intents. In explicit intents the name of the component is mentioned while in implicit intents an action string which can be composed of the required action and the related data, the component category and some extra fields to define different needed data are used—[12]. Based on the action string, Android sends implicit intents to the appropriate components by checking these action strings against the intent filters of components. Intent filters introduce the type of actions which are done by each component.

In order to have secure Inter-Component communication, components of an application can be protected by permission labels. For using services provided by the APIs, applications should declare the needed permissions for accessing the relevant components in the package manifest file at install time to be granted by the users [13].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

For instance if application A needs to use the service provided by a component of application B and this component is protected with a permission label, Application A should declare this permission in its manifest file in order to be granted by the user.

3. ANDROID SECURITY MECHANISMS

In common, different security mechanisms which are used in Android framework can be classified into three groups: Linux mechanisms, environmental features, and Android-specific mechanisms. Since the aim of this paper is to express the shortcomings in Android Specific security mechanism in order to address privilege escalation attacks, in the following, the classification of this mechanism introduced by Google: Component encapsulation, permission framework, and application signing [11, 14, 15] will be presented.

A. Component Encapsulation

An Android application can encapsulate its component. This mechanism prevents other applications from accessing to its components, because they have a separate userID. This is realized essentially through the definition of the "exported" property. If it is essential for a component of an application that is accessible either to its main application or other applications that bear the same user ID, therefore, the "exported" property should be "false"; in the same way, in order to be publicly available, the "exported" property should be "true". Developers should always keep in mind to set the "exported" property manually since the default value might not coincide with the required one.

B. Android Permission Framework

Android's permission mechanism applies limitations on particular operations that an application is able to perform. There are more than 100 built-in permissions that manage operations like using the Internet (INTERNET), taking pictures (CAMERA), making a phone call (CALL_PHONE), modify the current configuration (CHANGE_CONFIGURATION), and even disabling the phone permanently (BRICK). Also developers can define additional permissions in Android applications. In order to get permission, an application needs to request it explicitly at install time. Permissions have associated protection levels:

- Normal – application-level permissions which are not dangerous like turning on the phone's vibration, this kind of permission does not need user's confirmation;
- Dangerous – these are high-risk permissions that may provide access to the user private data or dangerous functionalities. Granting such permissions needs user's confirmation;
- Signature – these permissions are granted to applications with the same signature; and
- SignatureOrSystem – these permissions can be granted to packages installed in the Android system image.

As well as defending protected framework APIs, the permission mechanism is needed and should be applied in order to protect different components in an application [2].

C. Application Signing

The Android system requires that all installed applications be digitally signed. The signed apk is valid as long as its certificate is valid and the enclosed public key successfully verifies the signature [14]. There is no necessity to acquire these certificates from an authority. It allows Android application developer to self-sign the application. The Android only employs the certificate as a means of identifying the author of the application so that it will be able to launch reliable connections between applications.

4. SOME OF THE APPROACHES TO PRIVILEGE ESCALATION ATTACKS

Although Android integrates many security features to enhance the security of Smartphones, attackers use available security holes to perform their malicious actions. For example, component encapsulation may work best when developers take necessary measures to develop protected applications. However it is very much seen that there are many benign applications which are exploited by malicious applications due to their vulnerable interfaces. As another example, many vulnerabilities have been detected in the permission mechanism of Android that significantly raise the probability of installing malicious applications [2]. The Android permission mechanism is not enough

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

secured and gives malicious applications the opportunity of misusing permissions that may be granted by unaware users . Moreover, users are unable to approve only some of the permissions requested by an application. They can grant all or none of the requested permissions to an application and cannot verify that these permissions are used for benign purposes or not. In addition, shared user ID mechanism lets applications to share their permissions without users explicit approval.

Over the few years, many researchers in academia and industry proposed complicated extensions to fortify the Android's security framework. They mainly focus on protecting the user data and mitigating some types of privilege escalation attacks. In this section, some of the most well-known approaches are presented.

"Saint" [16] introduces a fine-grained access control extension that secures applications against being misused by malicious applications. Saint let application developers define access control policies to keep the components of their applications safe. These policies can determine the significant factors of calling applications like defining the permissions which are required by a caller application to access the components of callee. In this way the caller must have at least the same permissions that the callee has. With this method an application can specify applications which can access its interfaces. Saint mechanism trusts application developers who are not expert in security to consider the policies in their applications. Therefore it could be an error-prone approach which expects developers to consider proper policies.

"CRePE" [17] is an extension that tries to solve the problem of over-privileged applications by using context-related policies which bounds the privileges of an application or some of its functionalities by considering the contextual limits like time, geographical location and noise. The policies in CRePE can be defined by user as well as trusted third parties. For example these policies can be set by an organization for all the employees in a company. In CRePE contextual information are considered and appropriate policies are defined to restrict the privileges granted to an application. So it does not mitigate privilege escalation attacks.

"MockDroid" [13] is another extension for Android which prevents an application accessing the critical information and important resources. As it is inferred from its name, it makes users able to 'mock' an application's access to critical resources. There might be important resources and sensitive data on the phone that can be misused by some applications, therefore, MockDroid provides empty or fake information instead of real data when these applications request to access these resources. Since there is much sensitive information stored in Smartphone, and some of the applications use this information in an inappropriate way, MockDroid presents empty or bogus information whenever the application requests access. This solution empowers users to recall access to specific resources at run-time. This method enables users to opt between disclosing the sensitive data and functionality at the time of using an application. Whenever MochDroid serves an application with the fake information, user will be notified through providing some information.

"TISSA" [8] works very similar to MockDroid in controlling the access of unreliable applications to important resources and sensitive information. Unlike MockDroid which provides applications with the empty of fake data, TISSA presents true information but with low accuracy. The difference between TISSA and MockDroid is the quality of the data which is provided for the applications. This mechanism is able to make a balance between efficiency and flexibility. For instance, if an application needs the user location to provide weather information, TISSA provides the location data of some places near the user real place. This model challenges to come to a good balance between efficiency and flexibility. TISSA provides only one single privacy setting for one type of sensitive information. Sometimes it seems too coarse-grained.

"TaintDroid" [4] is an Android extension that tracks the flow of critical and personal data through untrusted applications. In TaintDroid, any third party application downloaded from the market is considered as an unreliable application. It monitors the way these applications access or change the sensitive data in users` devices. In this way TaintDroid can find out when and how the private data flow out of the phone through unreliable applications. The obtained information by this method helps users or other security services analyze the behavior of the applications. TaintDroid is capable of tracing data leakage and detecting suspicious actions through tracking the explicit data flows, though it is not able to detect private data leakage through implicit flows.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

"QUIRE" [20] with a low overhead mechanism prevents the unprotected interfaces of a benign application from being misused by a malicious app. By tracing RPC chains, QUIRE checks all the applications in the chain in order to find out they have the needed permissions to make an application call or not. The shortcoming of the Quire is that it is visible to the developers and also it cannot protect private data from disclosure to the remote servers through internet. Moreover this mechanism is not able to detect and prevent colluding attack

"The AppFence" [21] is similar to TISSA and MochDroid. It compounds two approaches in order to protect information disclosure by untrusted apps. First approach replaces the sensitive data with unreal information and the second prevents network transmissions that contains important data that are provided only for local activities by the users. AppFence, by taking advantage of TaintDroid provides users with another mechanism to mock the critical information in order to prevent the data disclosure. AppFence can do nothing to the problem of colluding attack. In this approach, mocking valid data allows only data anonymity and does not provide other alternative over private data.

"XManDroid" [11] is an Android extension for detecting and preventing privilege escalation attacks by examining and analyzing the communications between applications based on extended system policy. XManDroid proposes an efficient detection of covert channels which executed through the Android core services and content providers. This mechanism traces ICC traffic in order to detect those ICC calls that may cause privilege escalation according to the system policies. The proposed system track all the Inter Component Communication traffic and verify if an Inter Component Communication call can result escalating privileges based on appropriate system policy. Despite the previous capabilities XManDriod is not able to control communication channels executed outside the Inter Component Communication framework. Moreover, the lack of fine-grained access policies results in choosing between two options, all or no access to the resources and no intermediary approach is introduced.

"Porscha" [22] proposes policy-based secure content managing in Android. The main purpose of this approach is to associate any sensitive data or resources to a specific Smartphone and to a typical set of apps. Content sources (e.g. devices transmitting SMS, e-mails, etc.) can be binded to a Digital Rights Management policy in order to be protected. However, Porscha in some cases propose a more fine-grained solution, it is not able to inhibit disclosure of data which are not tainted with a security policy. Moreover, the primary aim of Porscha is to monitor and manage data flows (explicit flow), and privilege escalation attacks based on control flows (implicit flow) are not considered.

"ComDroid" [23] is a novel static analysis tool that identifies deficiencies in application interaction. It explores an application and identifies vulnerable interfaces and security-critical intent/broadcast transmissions. For example, it alerts the app developer about the potential attacks which result from sending private data through a public broadcast. In case of the existence of a malicious broadcast receiver, disclosing the data through eavesdropping the message can be possible and probable with a high percentage. However, this approach is capable of identifying application communications with deficiencies; it is not able to unfold privilege escalation attacks that are based on multiple colluding applications, regarding to the fact that it only concentrate on a single application. Moreover, normally static analysis tools are error-prone, since they are not able to completely foresee the real-time application communication.

"Kirin" [24] tries to solve the problem by checking the applications' permissions at install time. By using security requirements engineering and defining appropriate rules, Kirin detects the applications that has the potential to perform dangerous actions. These rules are composed of a combination of critical permissions that gives the application the ability of conducting malicious activities. Although, this mechanism can mitigate the threats caused by applications with excess permissions, it is not capable of detecting privilege escalation attack. This author also proposed a method [25] to decompile and analyze the source of applications to detect any possible way of leaking data.

"A Privilege Escalation Vulnerability Checking System" [26] considers applications which are not well protected and can be misused by malicious applications in order to conduct privilege escalation attack. In this research, they proposed a vulnerability checking system to verify if an application is vulnerable to privilege escalation attack.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Felt et al. [27] applies a type of static analysis to verify if an Android application is over-privileged or not. It examines all the permissions an application requests, and in case of not using requested permission, it concluded that the application is over-privileged. Felt et al. [28] also conducted a survey on applications on the Android Market to mark the applications that request dangerous permissions. None of above mentioned mechanism result to detect or categorize malicious applications.

Au et al. [29] conducts a survey on the permission system of the most popular Smartphone OSs and classifies them according to the level of control given to users, the amount and level of information they provide to users and the level of interactions they need from the user to serve properly. Moreover, they explained most problematic issues result from extracting permissions-based information from Android applications.

Vidas [30] proposes a mechanism that helps developers defining a minimum acceptable set of permissions needed for a particular mobile app. This mechanism is based on analyzing the code of the application and deducts the minimum acceptable set of permissions needed in order for the application to work properly.

Some of the most important works are summarized in the Table 1.

It may be worth to mention that while proposing extensions for Android security can be a useful paradigm for researchers, yet it might not providing end-users with applicable solutions which empower them best protect their device and private sensitive information.

5. CONCLUSION

In this paper, the types of attacks related to shortcomings of Android permission framework such as confused deputy attacks, collusion attacks and attacks result from applications with excessive privileges are discussed. It has been mentioned that users grant permissions to applications without knowing that whether they use critical services legitimately or not. Moreover, applications may share their permissions in order to perform actions beyond their normal permissions. Besides, there are many over-privileged but benign applications with vulnerable interfaces that are exploited by attackers. In the recent years many researches have been conducted to address different types of attacks related to shortcomings of Android permission framework. Most of them proposed some modification to Android framework in order to mitigate some deficiencies in it. However they address the problem by proposing effective solutions theoretically, still users cannot benefit from them.

REFRENCES:

[1] Au, K., Zhou, Y., Huang, Z., Gill, P., and Lie, D. (2011). Short paper: a Look at Smartphone Permission Models. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages. 17-21 October. Chicago, IL, USA, ACM: 63–68.

[2] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., and Dolev,S. (2009). Google Android: A state-of-the-Art Review of Security Mechanisms. CoRR, abs/0912.5101.

[3] Mahaffey, K., Hering, J. (2010). App Attack: Surviving the Explosive Growth of Mobile Apps.

[4] Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., and Sheth, A.N. (2010). TaintDroid: an Information-Flow Tracking System for Real-time Privacy Monitoring on Smartphones. *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. October. Vancouver, BC, Canada: OSDI'10, Article No. 1-6.

[5] Egele, M., Kruegel, C., Kirda, E., and Vigna, G. (2011). PiOS: Detecting Privacy Leaks in iOS Applications. *Proceedings of 18th Annual Network and Distributed System Security Symposium*. 6 – 9 February. San Diego, CA, USA: NDSS.

[6] Bradley, T. (2011, March 2). DroidDream Becomes Android Market Nightmare. PCWorld, Retrieved December 15, 2012, from http://www.pcworld.com/businesscenter/article/221247/droiddream_becomes_android_market_nightmare.html

[7] Thurm, S. and Kane, Y.I. (2010, December 17). Your Apps Are Watching You. The Wall Street Journal, Retrieved December 15, 2012, from http://online.wsj.com/article/SB10001424052748704694004576020083703574602.html

[8] Zhou, Y., Zhang, X., Jiang, X., and Freeh, V.W. (2011). Taming Information-Stealing Smartphone Applications (on Android). *Proceedings of the 4th International Conference on Trust and Trustworthy*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Computing. 22-24 June. Pittsburgh, PA, USA, TRUST: 93-107.

[9] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A.R., and Shastry, B. (2012). Towards Taming Privilege-Escalation Attacks on Android. *Proceedings of 19th Annual Network & Distributed System Security Symposium*. 5-8 February. San Diego, California, USA, NDSS.

[10] Android Security Overview, Security and Permissions. (2012). Retrieved December 15, 2012, from http://source.android.com/tech/security/#android-application-security

[11] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., and Sadeghi, A.R. (2011). Xmandroid: A new Android Evolution to Mitigate Privilege Escalation Attacks. Tecnical report, TR-2011-04, Technische Universität Darmstadt, Darmstadt, Germany.

[12] Nauman, M., Khan, S., and Zhang, X. (2010). Apex: Extending Android Permission Model and Enforcement with User-Defined Runtime Constraints. *Proceedings of the 5th ACM Symposium on Information*. 02 May. New York, USA: ACM, 328–332.

[13] Beresford, A.R., Rice, A., and Skehin, N. (2011). MockDroid: Trading Privacy for Application Functionality on Smartphones. *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, ser*. 1-2 March. New York, NY, USA: ACM, 49-54.

[14] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., and Glezer, C. (2010). Google android: A Comprehensive Security Assessment. *IEEE Security and Privacy*. 8 (2), 35–44.

[15] Davi, L., Dmitrienko, A., Sadeghi, A.R., and Winandy, M. (2010). Privilege Escalation Attacks on Android. *Proceedings of the 13th International Conference on Information Security*. 25-28 October. Boca Raton, FL, USA, ISC: 346–360.

[16] Ongtang, M., McLaughlin, S., Enck, W., and McDaniel, P. (2009). Semantically Rich Application-Centric Security in Android. *Proceedings of Proceedings of the 2009 Annual Computer Security Applications Conference*. December. Washington, DC, USA, ACSAC: pages 340-349.

[17] Conti, M., Nguyen, V.T.N, and Crispo, B. (2010). Crepe: Context Related Policy Enforcement for Android. *Proceedings of the 13th International Conference on Information Security*. 25 – 28 October. Boca Raton, FL, USA, ISC: 331– 345.

[18] Hering, J., Mahaffey, K., and Burgess, J. (2010, 27 July). Introducing the App Genome Project. LOOKOUT, Retrieved December 15, 2012, from http://blog.mylookout.com/2010/07/introducing-the-app-genome-project/

[19] Calo, R., Young, R., Smith, A., and Gelman, L. (2010). WhatApp. Retrieved May 6, 2012, from http://www.whatapp.org

[20] Dietz, M., Shekhar, S., Pisetsky, Y., Shu, A., and Wallach, D. S. (2011). Quire: Lightweight Provenance for Smartphone Operating Systems. *Proceedings of 20th USENIX Security Symposium*. 8 -12 August. San Francisco, CA, USA, USENIX.

[21] Hornyack, P., Han, S., Jung, J., Schechter, S., and Wetherall, D. (2011). These Aren't the Droids You're Looking for: retrofitting android to protect data from Imperious Applications. *Proceedings of the 18th ACM Conference on Computer and Communications Security*. 17 – 21 October. New York, USA: ACM, 639-652.

[22] Ongtang, M., Butler, K., and McDaniel, P. (2010). Porscha: Policy Oriented Secure Content Handling in Android. *Proceedings of the 26th Annual Computer Security Applications Conference*. 6-10 December. New York, NY, USA, ACM: 221-230.

[23] Chin, E., Felt, A.P., Greenwood, K., and Wagner, D. (2011). Analyzing Inter-Application Communication in Android. *Proceedings of 9th Annual International Conference on Mobile Systems, Applications, and Services*. 28 June – 1 July. New York, NY, USA: ACM, 239-252.

[24] Enck, W., Ongtang, M., and McDaniel, P. (2009). On Lightweight Mobile Phone Application. *Proceedings of the 16th ACM conference on Computer and Communications Security*. 9-13 November. Chicago, IL, ACM: 235 – 245.

[25] Enck, W., Octeau, D., McDaniel, P., and Chaudhuri, S. (2011). A Study of Android Application Security. *Proceedings of the 20th USENIX Conference on Security*. 8-12 August. San Francisco, CA, USA, SEC: 21–21.

[26] Chan, P.P.F, Hui, L.C.K, and Yiu, S.M. A Privilege Escalation Vulnerability Checking System for Android Applications. (2011). *Proceedings of 2011 IEEE 13th International Conference on Communication Technology*. 25-28 September. Jinan, China: IEEE, 681-686.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[27] Felt, A., Chin, E., Hanna, S., Song, D., and Wagner, D. (2011). Android Permissions Demystified. *Proceedings of the 18th ACM conference on Computer and Communications Security*. 17 – 21 October. New York, USA, ACM: 627–638.

[28] Felt, A., Greenwood, K., and Wagner, D. (2011). The Effectiveness of Application Permissions. *Proceedings of the 2nd USENIX Conference on Web Application Development*. 15 – 16 June. Portland, OR, USA: 83-94.

[29] Au, K., Zhou, Y., Huang, Z., Gill, P., and Lie, D. (2011). Short paper: a Look at Smartphone Permission Models. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages. 17-21 October. Chicago, IL, USA, ACM: 63–68.

[30] Vidas, T., Christin, N., and Cranor, L. (2011). Curbing Android Permission Creep. *Proceedings of Web 2.0 Security and Privacy Workshop*. 26 May. Oakland, California, USA, W2SP: 2.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1. Comparison Table of the Current Approaches

Model	Security mechanism	mitigate			Main deficiencies
		Excessive permissions attacks	Colluding attack	Confused deputy	
Saint	Declares access control rules to protect the components of applications	✓		✓	Relies on developers to consider saint policies in implementing their applications
CRePE	Develops context-related policies to limits an application's privilege	✓			Has nothing to do with mitigating privilege escalation attacks
MockDroid	Prevents user data by `mock' an application's access to critical resources	✓			Has nothing to do with mitigating privilege escalation attacks
TISSA	Provides suspicious applications accurate but low fidelity data	✓			Has nothing to do with mitigating privilege escalation attacks
TaintDroid	Detects when and in what way personal data leaves the phone through unreliable apps	✓			- Only traces explicit flows and does not consider implicit flows - Has nothing to do with mitigating privilege escalation attacks
Quire	Approaches the problem through tracking RPC chains			✓	- Is not invisible to app developer - Cannot identify and impede colluding attack
AppFence	Proposes two privacy controls to enable users to protect their data	✓			Has nothing to do with mitigating privilege escalation attacks
XManDroid	Inhibits privilege escalation attacks at runtime by analyzing of interaction among applications and according to extended system policy.		✓	✓	Is incapable of controlling communication channels executed outside the ICC framework
Kirin	Uses policies to detect the applications with inappropriate infrastructure	✓			Is unable to mitigate colluding and privilege escalation attacks
Checking Privilege Escalation	Proposes a vulnerability checking system to check if an application can be potentially leveraged by an attacker to launch such privilege escalation attack.			✓	Fails to prevent the excessive permission and colluding attacks