Enhanced Permission based Scheme for Android Malware Detection

Dhavalkumar K. Baraiya[1] Prof. Hiteishi Diwanji[2]
[1]M.E IT–Research Scholar [2]Associate Professor
[1,2]Department of Information Technology
[1,2]L. D. College of Engineering, Ahmedabad

Abstract— With the increased use of Android smartphones, more Android malwares are being developed. Considering the wide range of functionalities provided by smartphones, smartphone-users' privacy and security of their personal information is of vital importance. Android has been the most targeted platform for malware attacks, due to being the most popular platform, open source system and its vulnerable architecture. In this paper, we have studied about android malware detection, android permission mechanism, collusion attack, and presented Enhanced Permission Based Scheme for Android Malware Detection. Our experimental results show that it has improved the accuracy of detection of malwares that attempt to evade detection. Consequently, it will also help in mitigating application collusion attacks.

Key words: Android Malware, Mobile Security, Malware Detection, Collusion Attack, Permission Based Scheme

I. INTRODUCTION

The worldwide smartphone market grew 27.2% in third quarter of 2014[1]. Android shipments lead the global smartphone market, with 283 million units shipped and over 84% of the market share in the third quarter of 2014. According to the Kaspersky Security Bulletin 2012[2], 99% of newly discovered mobile malware target the Android platform, with a very small amount targeting Java- and Symbian-based smartphones. Android remained the prime target for malicious attacks in 2013 as well. 98.05% of all malware detected targeted Android, which implies adequate vulnerability of its architecture and its popularity. "Malware[3] can be defined as any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system." e.g., viruses, worms, and Trojan horses.

Malware detectors are used to detect and remove malware. Malware detectors take two inputs:

1) Knowledge of the malicious behavior
2) Program under inspection

Malware Detection Techniques can be categorized broadly into two categories[3].

1) Anomaly-based detection
2) Signature-based detection

A. *Anomaly-based Detection*

- This technique uses its knowledge of what constitutes normal behavior to decide if the program under inspection is malware.
- A special type of anomaly-based detection is referred to as Specification-based detection.
- Specification-based techniques leverage some specification or rule set of what is valid behavior in order to decide if the program under inspection is malware. Programs violating the specification are considered anomalous and usually, malicious.

B. *Signature-based Detection*

- This technique uses its characterization of what is known to be malicious to decide if the program under inspection is malware.
- Its effectiveness depends on the signature of the malicious behavior.

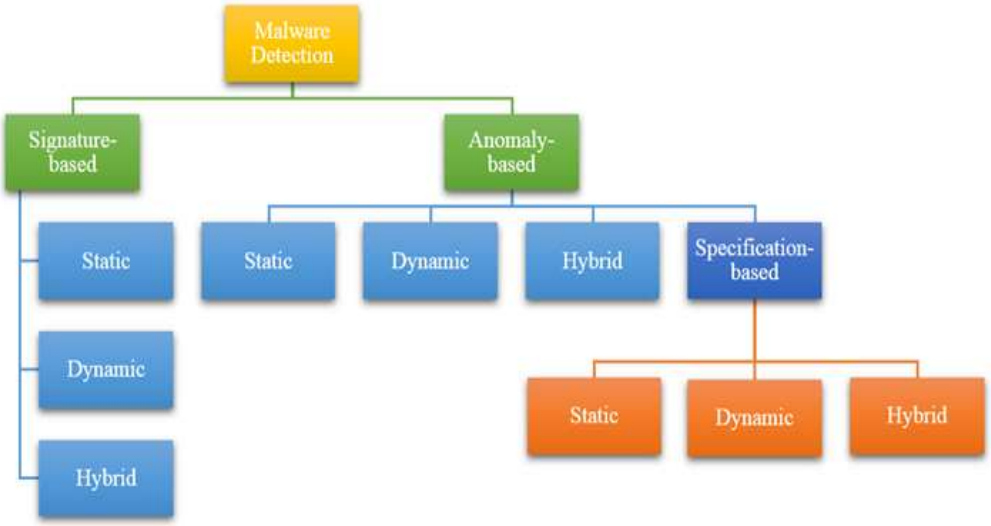


Fig. 1: Types Of Malware Detection Techniques

As shown in Fig. 1, Malware detection techniques can employ one of three different approaches:

- Static
- Dynamic
- Hybrid

Table 1 summarizes the Comparison of different approaches for malware detection:

Approach	Features Analyzed	Advantages	Disadvantages
Static	− It uses Syntax or structural properties of the program under inspection. e.g., a static approach[4] to signature-based detection would only leverage structural information to determine its maliciousness.	− It allows a complete analysis of a given program. − It can cover all possible execution paths of a malware sample. − It is generally safer than dynamic approach as the source code is not actually executed.	− It is ineffective against previously unseen attacks and hence it cannot detect new and unknown intrusion methods as no signatures are available for such attacks. − The source code of malware samples is not readily available − It can be extremely time-consuming and awkward process
Dynamic	− It will leverage runtime information of the Process Under Inspection. e.g., systems seen on the runtime stack − It attempts to detect malicious behavior during program execution or after program execution.	− It can avoid obfuscation issues, so it is easy to see the actual behavior of a program. − It can detect new intrusion method and can detect new malware	− The main drawback is that usually it monitors only one execution path, so it suffers from incomplete code coverage. − There is also the danger of harming third party systems, if the analysis environment is not properly isolated or restricted respectively. − Furthermore, malware samples may alter their behavior or stop executing at all once they detect to be executed within a controlled analysis environment.
Hybrid	− It combines the above two approaches. − It uses both static and dynamic information to detect malware.	− It utilizes advantages and reduces disadvantages of each of the static and dynamic approaches.	

Table 1: Comparison of Different Approaches for Malware Detection

II. ANDROID PERMISSION MECHANISM

Android system has a strict permissions management mechanism to restrict the behavior of applications[5]. Android OS provides around 75 Permissions that have Medium control, High information, and Low interactivity[6]. These permissions manage operations like making a phone call (CALL_PHONE), using the Internet (INTERNET), taking pictures (CAMERA), and even disabling the phone permanently (BRICK)! Some system functions are not allowed to be called by default when an android application is running, such as phone calls, SMS, Bluetooth, WIFI, etc. These functions generally correspond to the specific hardware devices. In order to use these functions, corresponding permissions must be granted to the program using the <uses-permission> tag in AndroidManifest.xml.

Android permissions have their associated protection levels as follows[7]:

1) Normal – Application-level permissions which are not dangerous (e.g., turning on the phone's vibration). It does not need user's confirmation.
2) 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.
3) Signature – These permissions are granted to applications with the same signature.
4) Signature Or System – These permissions can be granted to packages installed in the Android system image.

As shown in Fig. 2, when installing a new Android application, Android system prompts the user for granting access to the permissions requested by the application, and Users can choose refusing to install if there are sensitive permissions. At run-time the operating system enforces that the application doesn't access any data or resource for which it does not have a permission.

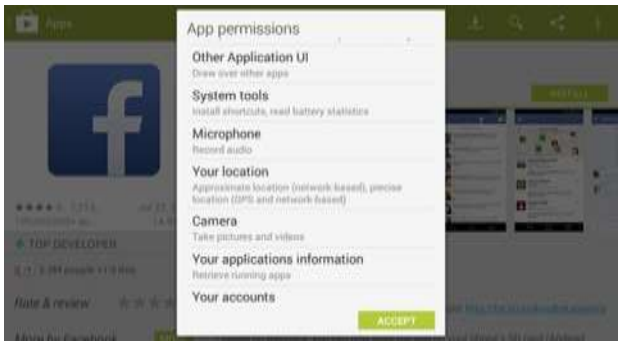


Fig. 2: Android App Permissions Prompt – Screenshot

III. APPLICATION COLLUSION ATTACK

We consider our smartphones as a trusted, private channel of communication, and it's widely used to receive authentication information to access specific online services[8]. If the smartphone is infected by a malware, this personal information can be leaked or abused by colluding applications. Currently almost all academic and industry efforts are focusing on single malicious apps, and almost no attention has been given to such colluding apps[9]. Even existing antivirus softwares cannot detect collusion attacks. We have focused on Android apps, as Android is an open-source system and it has adequate vulnerabilities.

Android security mechanism relies on restricting apps by combining digital signatures, sandboxing, and permissions. However, these restrictions can be bypassed, without user's consent/ knowledge, by colluding apps whose combined permissions allow them to carry out attacks that neither app could carry out alone. For example, one app would request permission to access personal data, but it would not request permission to access Internet. In fact, it will secretly communicate with another application(s) that has Internet access, without user's consent/ knowledge. This way the app with access to personal data, would then send that information to the app with Internet access, which would subsequently transmit it to a destined attacker. Thus, collusion attacks allow apps to indirectly execute operations, which they should not be able to execute, based on their declared permissions.

According to the EPSRC's announcement, this is a relatively little-studied attack model[10]. Collusion attacks can be hard to detect. Existing security software that are used for the analysis of application permissions, analyze and report the permissions independently, for each individual application, and therefore do not take into consideration application collusion. Collusion attacks are a consequence of per-app permission models.

Collusion attack is a really sophisticated malware attack. Smartphone users are not informed about possible implications of application collusion attacks[5], instead, they are indirectly made to believe that by approving the installation of each app independently, they can limit the damage that an app can cause. Users should be careful about apps they are installing, particularly if downloading from an unofficial app store. You'd be cautious if an app is requesting access to lots of irrelevant permissions, e.g., a simple Unit convertor app would never need access to contacts information on your phone, and though, if such an app is requesting access to contact information, then that app would be having some malicious intents.

Colluding applications are those applications that cooperate in a violation of some security property of the system[11]. Such applications do not need to individually exploit software vulnerabilities or break any security permission. Instead of that, they use existing or create new communication channels to perform actions or access resources to which they would independently be unable to perform, e.g., due to their respective permission restrictions. The attack by colluding applications is possible because current security mechanisms are not focused on controlling the channels that two applications can use to communicate. Instead, most efforts have been made to achieve application containment or sandboxing. The way in which the permission-based security model is used in Android, assumes users can correctly judge the implications of permissions that applications request. Although this might not be true for some users, increasing numbers of users do understand the risks of different types of access (e.g., to their personal data and to the network). If multiple Android apps are signed by the same certificate, then one app can expose its some functionality to another app, which may allow malicious authors to obfuscate their true (malicious) intent across multiple benign applications.

As shown in the Fig. 3, the ContactsManager or PasswordsManager application on the left and the Weather application on the right are colluding through a covert communication channel.

- The ContactsManager does not have access to the network, but has access to user's contacts.
- The Weather application has no access to user's contacts but can access the network.
- The ContactsManager leaks user's contacts to the Weather application, which then sends this information to a destined attacker.

When colluding applications are used for data leakage we classify them either as "sources", denoting a class of applications that has a permission to access private data, or as "sinks", representing a class of applications that can receive and forward the data to third parties. Fig. 3 depicts two colluding "source" and "sink" applications.

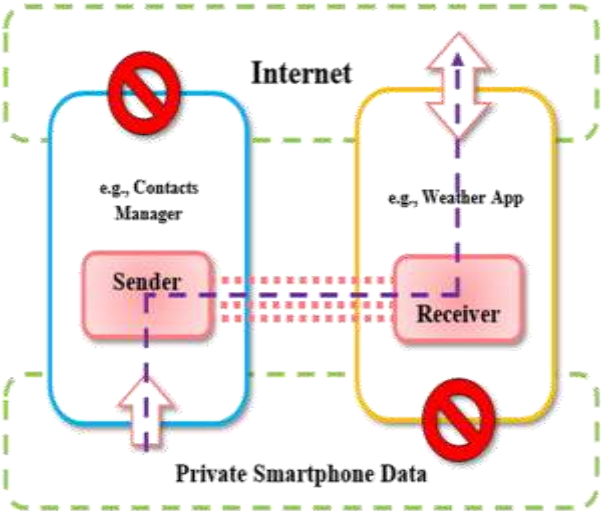
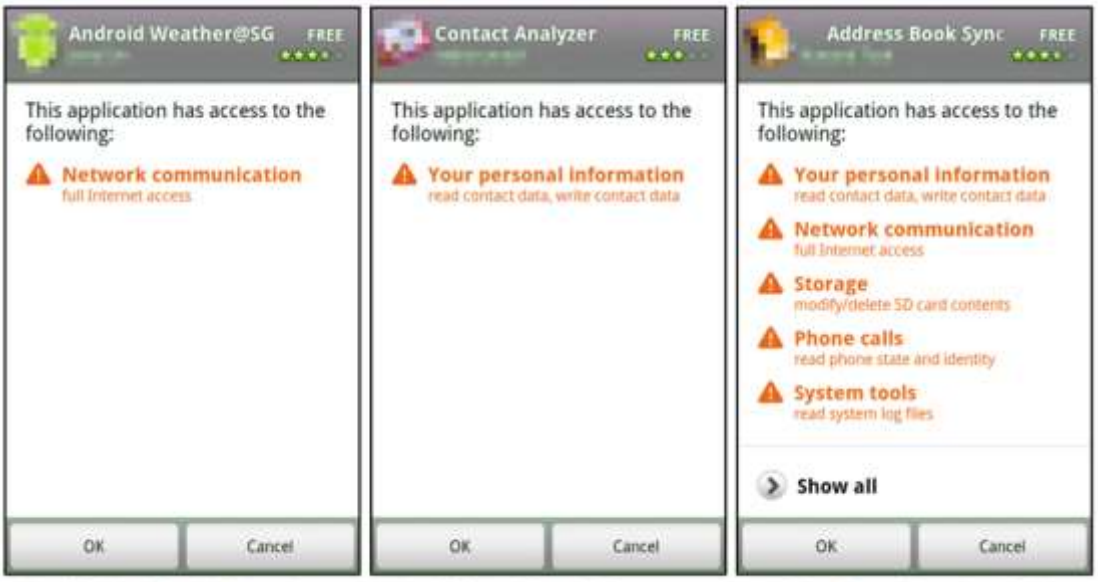


Fig. 3: Application Collusion Attack

Fig. 4 shows the Android permissions of three applications, presented to the user at their installation time. Here, the permissions of applications (a) and (b) should not

seem suspicious to the user as each application only request either access to personal data or to the network. However, Permissions of application (c), seem to be suspicious, as they require access to both the network and to the user's private data. So the user would refuse to install application (c) whereas s/he would feel that it is safe to install applications (a) and (b), although these applications may collude and disclose the users' private data.



(a) Application with only Network access. (b) Application with only Contact access. (c) Possibly suspicious Application.

Fig. 4[11]: The Screens Presented To the Users at Application Installation Time

IV. Enhanced Permission Based Scheme

In order to detect malware that evades detection by obfuscating their true intent across multiple benign applications, we will treat the applications signed by same certificate as single application-group with their combined requested permissions.

So, first of all we will check if multiple applications are signed with the same certificate.

Then, we'll compare those suspicious applications' permission-requests with the permission-requests that are most frequently requested by malwares but rarely by benign applications. Algorithm steps are as follows:

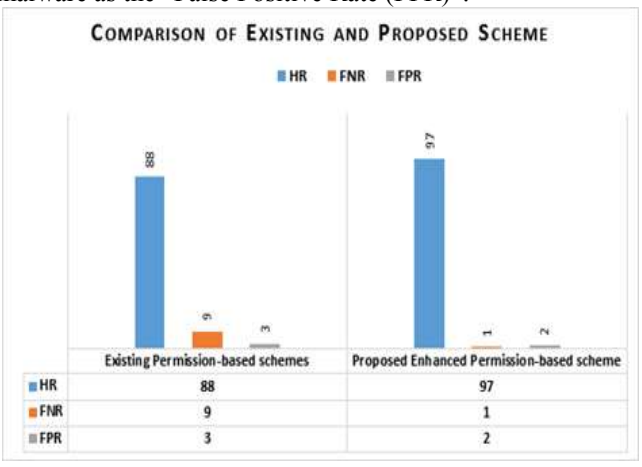
1) Step 1: Fetch certificate information of all APK files.
2) Step 2: Detect applications which are signed by the same certificate, and consider them as group(s) of suspicious applications.
3) Step 3: For every group of suspicious applications, consider all of their permission-requests to be requested by a single application.
4) Step 4: Now compare those suspicious applications' permission-requests with the permission-requests that are most frequently requested by malwares but rarely by benign Android applications.
5) Step 5: Based on the above analysis, Deduce which applications are benign and which are malware.

V. Results

Our experimental results show that existing permission-based Android malware detection schemes analyze permissions for each individual application, independently and hence fail to detect colluding applications, while our Enhanced permission-based scheme can detect such colluding applications by combining all the permissions requested by the applications signed by the same certificate and analyzing them against sensitive permission-combinations.

Now, taking into the consideration - the set of analyzed applications, the results ascertain that, at present, application collusion techniques are not often employed by applications available on the Android Market. Nevertheless, the potential of such collusion attacks is there: 2.5% applications could act as sources and 43.9% could act as sinks!

Based on our experiments, Fig. 4 clearly illustrates comparative analysis of existing schemes and proposed scheme for Android malware detection. In this chart, we have represented the percentage of malware detected as the "Hit Rate (HR)", the percentage of malwares that are not detected as the "False Negative Rate (FNR)", and the percentage of benign applications that are flagged as malware as the "False Positive Rate (FPR)".



VI. Related Work

We have provided an enhancement to existing permission-based Android malware detection schemes, now to ascertain that applications are indeed colluding, the suspected application-group(s)' behavior could be analyzed to detect malicious operations.

In order to mitigate collusion attacks, either applications' access to private information could be reduced or communicating possibilities among installed applications could be restricted properly.

Overt and covert communication channels should be mitigated through enforcing security policies[12] and careful analysis of the usage of visible and alterable variables used by system calls or using a formal model for analyzing programs using a semi-automated technique.

VII. CONCLUSION

In this paper, we have studied about android malware detection, android permission mechanism, collusion attack, and presented Enhanced Permission Based Scheme for Android Malware Detection. Android permissions, as an important security identity system, can also be meaningful for identifying Android malware and benign applications.

In our enhanced permission-based scheme, we have treated applications signed by the same certificate as single application-group with their combined permission-requests and analyzed them against sensitive permission-combinations. It will improve the accuracy of permission-based detection of malware that attempt to evade detection. Consequently, it will also help in mitigating collusion attacks.

REFERENCES

[1] "Smartphone OS Market Share, Q3 2014" http://www.idc.com/prodserv/smartphone-os-market-share.jsp
[2] "99% of all mobile threats target Android devices" http://www.kaspersky.com/about/news/virus/2013/99_of_all_mobile_threats_target_Android_devices
[3] Nwokedi Idika, Aditya P. Mathur – "A Survey of Malware Detection Techniques"
[4] Savan Gadhiya, Kaushal Bhavsar – "Techniques for Malware Analysis", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, Issue 4, April 2013
[5] Claudio Marforio, Hubert Ritzdorf, Aurélien Francillon, Srdjan Capkun - "Analysis of the Communication between Colluding Applications on Modern Smartphones", ACSAC '12 Dec. 3-7, 2012, Orlando, Florida USA
[6] Guillermo Suarez -Tangil, Juan E. Tapiador, Pedro Peris - Lopez , and Arturo Ribagorda, "Evolution, Detection and Analysis of Malware for Smart Devices", 1553-877X/14/$31.00 © 2014 IEEE
[7] Iman Kashefi, Mazleena Salleh - "A survey on mitigating attacks related to shortcomings of android permission framework", JATIT Vol. 55 No. 2
[8] Lorenzo Cavallaro, lecturer in the Information Security Group at Royal Holloway University of London http://eandt.theiet.org/news/2014/feb/colluding-apps.cfm
[9] Professor Tom Chen, a leading researcher at City University London
[10] http://www.epsrc.ac.uk/NEWSEVENTS/NEWS/2014/Pages/appattackers.aspx
[11] Claudio Marforio, Aurelien Francillon, Srdjan Capkun - "Application Collusion Attack on the Permission-Based Security Model and its Implications for Modern Smartphone Systems"
[12] William Enck, Peter Gilbert, Byung gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. – "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones", In Proceedings of the 9th USENIX Symposium on OSDI, 2010

