Palliates the Attack by Hacker of Android Application through UID and Antimalware Cloud Computing

Abu Sarwar Zamani[1], Sultan Ahmad[2*], Mohammed Yousuf Uddin[3], Asrar Ahmad Ansari[4], Shagufta Akhtar[5]

a.zamani@psau.edu.sa, s.alisher@psau.edu.sa, m.yousuf@psau.edu.sa, aaansari@ksu.edu.sa, shagufu@gmail.com

[1]Department of Computer and Self Development, Preparatory Year Deanship, Prince Sattam bin Abdulaziz University, Al-Kharj, Saudi Arabia

[2]Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia

[3]Department of Information Systems, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj, Saudi Arabia

[4]Researcher/E-Learning Consultant Medical Education Department, College of Medicine, King Saud University, Riyadh, KSA

[5]Dept. of Computer Science, Institute of Science & Information Technology, Chapra, Bihar, India

[*]Corresponding Author: Sultan Ahmad

Summary

The market for smart phones has been booming in the past few years. There are now over 400,000 applications on the Android market. Over 10 billion Android applications have been downloaded from the Android market. Due to the Android popularity, there are now a large number of malicious vendors targeting the platform. Many honest end users are being successfully hacked on a regular basis. In this work, a cloud based reputation security model has been proposed as a solution which greatly mitigates the malicious attacks targeting the Android market. Our security solution takes advantage of the fact that each application in the android platform is assigned a unique user id (UID). Our solution stores the reputation of Android applications in an anti-malware providers' cloud (AM Cloud). The experimental results witness that the proposed model could well identify the reputation index of a given application and hence its potential of being risky or not.

Keywords: *Smart phones; Android OS; Reputation based security; Inter Process Communication*

1. Introduction

Access control lists (ACLs) and permission-based security models allow administrators and operating systems to restrict actions on specific resources. In practice, designing and configuring ACLs (particularly those with a large number of configuration parameters) is a complicated task. More specifically, reaching a balance between the detailed expressiveness of permissions and the usability of the system is not trivial, especially when a system will be used by novices and experts alike. One of the main problems with ACLs and permission models in general is that they are typically not designed by the users who will ultimately use the system, but rather by developers or administrators who may not always for see all possible use cases. While some argue that the problem with these permission-based systems is that they are not designed with usability in mind [11], we believe that in addition to the usability concerns, there is not a clear understanding of how these systems are used in practice, leading security experts to blindly attempt to make them better without knowing where to start. While there are many widely deployed systems which use permissions, we focus on the empirical analysis of the permission model included in Android OS [1]. Android is a newcomer to the smart phone industry and in just a few years of existence has managed to obtain significant media attention, market share, and developer base. Android uses ACLs extensively to mediate inter-process communication (IPC) and to control access to special functionality on the device (e.g., GPS receiver, text messages, vibrator, etc.). Android developers must request permission to use these special features in a standard format which is parsed at install time. The OS is then responsible for allowing or denying use of specific resources at run time. The permission model used in Android has many advantages and can be effective in preventing malware while also informing users what applications are capable of doing once installed.

The main objectives of our empirical analysis are: (1) to investigate how the permission-based system in Android is used in practice (e.g., whether the design expectations meet the real-world usage characteristics and (2) to identify the strengths and limitations of the current implementation. We believe such analysis can reveal interesting usage patterns, particularly when the permission-based system is being used by a wide spectrum of users with varying degrees of expertise.

2. Background

Access control systems have existed for a long time [17]. In its basic form, a security system based on access control lists allows a subject to perform an action (e.g., read, write, run) on an object (e.g., a file) only if the subject has been

Manuscript received August 5, 2021
Manuscript revised August 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.8.24

assigned the necessary permissions. Permissions are usually defined ahead of time by an administrator or the object's owner. Basic file system permissions on POSIX-compliant systems [12] are the traditional example of ACL-based security since objects – in this case, files can be read, written or executed either by the owner of the file, users in the same group as the owner, and/or everyone else. More sophisticated ACL-based systems allow the specification of a complex policy to control more parameters of how an object can be accessed. We use the term permission-based security to refer to a subset of ACL-based systems in which the action doesn't change (i.e., there is only one possible action to allow or deny on an object). This would be similar to having multiple ACLs per object, where each ACL only restricts access to one action. We note that reducing the allowable actions to one does not necessarily make the system easier to understand or configure. For example, in the Android permission model, developers implement finer level granularity by defining separate permissions for read and write actions.

2.1 Permission-Based Security Examples

An example of a permission-based security model is Google's Android OS for mobile devices. Android requires that developers declare in a manifest a list of permissions which the user must accept prior to installing an application. Android uses this permission model to restrict access to advanced or dangerous functionality on the device [14]. The user decides whether or not to allow an application to be installed based on the list of permissions included by the developer. Similar to Android OS, the Google Chrome web browser uses a permission-based architecture in its extension system [4]. Extension developers create a manifest where specific functionality (e.g., reading bookmarks, opening tabs, contacting specific domains) required by the extension can be requested. The manifest is read at extension install time to better inform the user of what the extension is capable of doing, and reduce the privileges that extensions are given [10]. In contrast, Firefox extensions, which do not have this permission architecture, run all extension code with the same OS-level privileges as the browser itself. A third example of a currently deployed permission-based architecture is the Blackberry platform from Research in Motion (RIM). Blackberry applications written in Java must be cryptographically signed in order to gain access to advanced functionality (known as Blackberry APIs with controlled access) such as reading phone logs, making phone calls or modifying system settings [3].

2.2 Related Work

The design and implementation of a framework to detect potentially malicious applications based on permissions requested by Android applications. The framework reads the declared permissions of an application at install time and compares it against a set of rules deemed to represent dangerous behaviour. For example, an application that requests access to reading phone state, record audio from the microphone, and access to the Internet could send recorded phone conversations to a remote location. The framework enables applications that don't declare (known) dangerous permission combinations to be installed automatically, and defers the authorization to install applications that do to the user.

Ontang et al. [18] present a fine-grained access control policy infrastructure for protecting applications. Their proposal extends the current Android permission model by allowing permission statements to express more detail. For example, rather than simply allowing an application to send IPC messages to another based on permission labels, context can be added to specify requirements for configurations or software versions. The authors highlight that there are real-world use cases for a more complex policy language, particularly because untrusted third-party applications frequently interact on Android. On the topic of analysis of permission-based architectures.

3. Proposed Solution

As part of a solution to the above identified pitfalls in the android security model, we propose a reputation based security trust model to evaluate and validate the applications prior to installation. We have also analysed the consequences of a malicious application that has managed to get installed with the full consent of the end user. The Internet is full of genuine and malicious applications. An Android mobile owner can download different applications with varying reputation ratings. In this model, it is proposed that after downloading and before installing, the mobile device asks the AM Cloud for the reputation of the downloaded application.

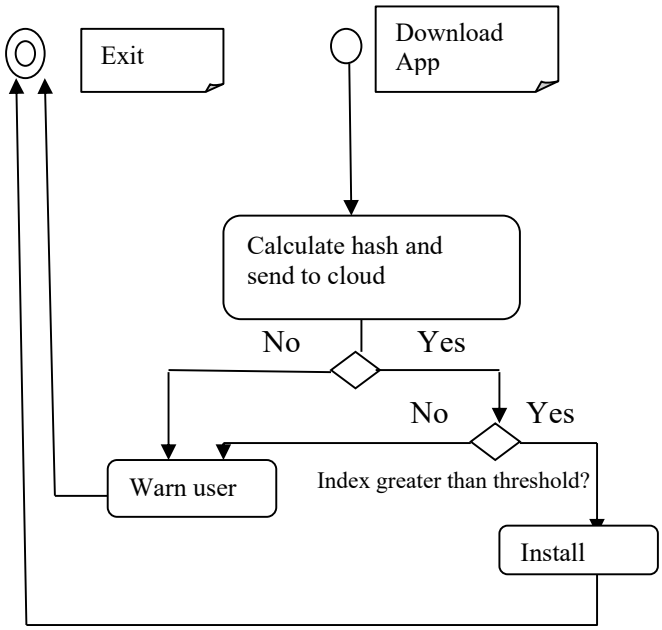

Figure-1: overview of the proposed protocol

Based on the downloaded applications' behaviour and reputation index the downloaded application can be classified in any of the following three ways.

A. The application has built a good reputation and there is likely no harm installing it on the client's device. Good reputation will be set after some threshold of positive feedback from those clients that have downloaded and automatically reported.
B. The application has not yet developed any good or bad reputation in the AM Cloud. In general, if an application has not developed a good reputation, we should be extremely cautious with such an unknown application. In this scenario, the anti-malware provider may wish to recommend that the user does not install the application or that the user installs the application in a sandbox.
C. The application has a bad reputation. In this case, the user is warned about the application's bad reputation.

4. Experiments

Concerning just the applications which have not yet developed a strong reputation, we need to analyse those applications. To analyse the behaviour of an Android application, it is easier to start with analysing the set of permissions that the application has set in the Android application package file which includes all of the application's code, resources, assets, and manifest file. To do this, we have experimented with a reputation based security model for Android applications. A second experiment was also done to analyse how a malicious application could track a mobile owners' location and report it to a third party. The results were achieved using two experiments.

4.1. Experiment-1

One solution which has been used by anti-malware vendors is to perform analysis of the application, on the Android platform. However the Android is low on resources, such as performance, battery life and main memory. So it makes more sense to perform the analysis in the AM Cloud. To overcome these issues, another solution which has been used by anti-malware providers is to upload the entire application for analysis (for each user). For our solution, we will minimize the uploading of applications to the AM Cloud. I.e., we do not want two users, with the same exact application, to both upload the same application. Our approach to minimize the uploading of applications now follows.

4.2. Experiment-2

In this second experiment, we have developed two applications namely Location Tracker, The Location Tracker application has ACCESS_FINE_LOCATION, ACCESS_MOCK_ LOCATION,and ACCESS_COARSE_LOCATION permissions in the user permission manifest file of the application. The manifest file declares which permissions the application must have in order to access protected parts of the API and interact with other applications [18]. It also declares the permissions that others are required to have in order to interact with the application's components [18]. The Location Tracker application implements a location listener class that returns the latitude and longitude of the present location by consulting the Location Manager, which provides access to the system location services. We can use the latitude and longitude to locate the associated geographic place such as the street address, hotel, and zip codes.

5. Further Discussion

Designing a permission-based system is a challenging task because system designers must anticipate what usage will be given to the permissions defined in their system. The analysis in this paper has helped to identify developer usage patterns in a real-world dataset of top Android applications. Additionally, there is a constant struggle to make the system highly configurable under different use-cases while maintaining a low level of complexity. Understanding how the permission model is used in practice can help in making modifications to improve currently deployed permission systems. Furthermore, our analysis shows correlations

between several of the infrequently used permissions. We note that having finer-grained permissions in a permission-based system enables users to have detailed control over what actions are allowed to take place. Whether it is beneficial to provide finer granularity will depend on many factors within a particular environment, as it increases complexity and thus may have, for example, usability impacts on designers and end-users. In the case of Android, having 'too many' permissions impacts both developers and end users. Developers must understand which permissions are needed to perform certain actions; determining this is often non-trivial, even for 'experts'. While some enthusiastic developers might take the time to learn what each of the 110 or more permissions do and request them appropriately when needed, other developers might choose to simply over-request functionality to make sure their application works.

5.1 Possible Enhancements to Android

The Android permission model does not currently make use of the implied hierarchy in its namespace. For example, a.p.SEND_SMS and a.p.WRITE_SMS are two independent permission labels, instead of being grouped, for instance, under a.p.SMS. Android includes an optional logical permission grouping [9] that is used for displaying permissions with more understandable names (e.g., one of the groupings reads "Services that cost you money" in-stead of a.p.CALL_PHONE). This grouping, however, does not allow developers to hierarchically define permissions, which could potentially extend current Android-defined permissions to express more detailed functionality. In the case of Android particularly, a permission hierarchy would allow for an extensible naming convention and help developers more accurately select (only the) needed features. One example would be a free application that displays ads from domains belonging to Admob. Currently a developer would include the ad code snippet, and request the a.p.INTERNET permission. This permission allows the application to communicate over any network and retrieve any data from any server in the world. A more fine grained hierarchical permission scheme could enable the developer to request the a.p.INTERNET. ADVERTISING (.admob.com) permission which could limit network connectivity to only download ads in static HTML from sub domains of Admob. A hierarchical permission scheme could help users understand why an application is requesting specific permissions, but more importantly, could help developer's better use the principle of least privilege. This modification is not backwards compatible with the currently deployed Android OS, therefore it might be better suited for an entirely new model instead.

5.2 Applicability to Other Permission-Based Systems

The methodology presented in this work has allowed us to understand how developers use the permission-based security model in Android. We believe that our methodology is applicable to explore usage trends in other permission based-based systems. A base requirement for the methodology to work is being able to display applications and associated permissions for this representation to be possible, the set of permissions requested by an application must be accessible. In the case of Android, the set is statically readable in a manifest, but other systems might have different implementations. Google's Chrome OS extension system [4, 10] uses an Android-like manifest and permissions to access advanced functionality, which makes this system a prime candidate for applying our methodology. An empirical study of a large set of third-party extensions using our SOM-based methodology could help identify what correlations, if any, are present in requesting permissions to open tabs, read bookmarks, etc. This may also be of use in addressing other security concerns raised in recent work [10].

6. Conclusion

We have introduced a methodology to the security community for the empirical analysis of permission-based security models. In particular, we analysed the Android permission model to investigate how it is used in practice and to determine its strengths and weaknesses. The Self-Organizing Map (SOM) algorithm is employed, which allows for a 2-dimensional visualization of highly dimensional data. SOM also supports component planes analysis which can reveal interesting usage patterns. We have analysed the use of Android permissions in a real-world dataset of 1,100 applications, focusing on the top 50 application from 22 categories in the Android market. The results show that a small subset of the permissions is used very frequently where large subsets of permissions were used by very few applications. We suggest that the frequently used permissions, specifically a.p.INTERNET, do not provide sufficient expressiveness and hence may benefit from being divided into sub-categories, perhaps in a hierarchical manner. Conversely, infrequent permissions such as the self-defined and the complementary permissions (e.g., install/ uninstall) could be collapsed into a general category. Providing finer granularity for frequent permissions and combining the infrequent permissions can enhance the expressiveness of the permission model without increasing the complexity (i.e., maintaining a constant over all permission count) as a result of the additional permissions. We hope that our SOM-based methodology, including visualization, is of use to others exploring independent permission-based models.

Acknowledgment

The authors would like to thank the Deanship of Scientific Research at Prince Sattam Bin Abdulaziz University, Alkharj, Saudi Arabia for the assistance.

References

[1] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google Android: A Comprehensive Security Assessment. In IEEE Security & Privacy, Volume 8, Issue 2, pp. 35–44, March–April 2010.

[2] T. Bläsing, L. Batyuk, A.-D. Schmidt, S.A. Camtepe and S. Albayrak. An Android Application Sandbox system for suspicious software detection. In Proceedings of 5th International Conference on Malicious and Unwanted Software (MALWARE 2010), Nancy, France, Oct. 19–20, 2010.

[3] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. Semantically Rich Application-Centric Security in Android. In Proceedings of the Annual Computer Security Applications Conference (ACSAC '09), Austin, TX, USA, December 6–10, 2009.

[4] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka. Towards Formal Analysis of the Permission-Based Security Model for Android. In Proceedings of Fifth International Conference on Wireless and Mobile Communications (ICWMC '09), Cannes/La Boca, France, August 23–29, 2009.

[5] P. Teufl, C. Orthacker, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder and O. Prevenhueber. Android Market Analysis with Activation Patterns, In Proceedings of 3rd International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MOBISEC 2011), Aalborg, Denmark, May 17–19, 2011.

[6] C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber. Android Security Permissions - Can we trust them? In Proceedings of 3rd International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MOBISEC 2011), Aalborg, Denmark, May 17–19, 2011.

[7] J. Burns. Developing Secure Mobile Applications for Android—An Introduction to Making Secure Android Applications, http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf, Accessed on May 8, 2012.

[8] E. Chin, A. Porter Feltm, K. Greenwood, and D. Wagner. Analysing the Inter-application Communication in Android, University of California, Berkeley, Berkeley, CA, USA.

[9] T. Vidas, D. Votipka, and N. Christin. All Your Droid Are Belong To Us: A Survey of Current Android Attacks, INI/CyLab, Carnegie Mellon University.

[10] Android Market, http://www.android.com/market, Accessed on May 13, 2012.

[11] Android permissions, http://android.git.kernel.org/?p=platform/frameworks/base.git;a=blob;f=core/res/AndroidManifest.xml. Accessed on May 13, 2012.

[12] A. Shabtai, Y. Fledel, and Y. Elovici. Securing Android-powered mobile devices using SE Linux. In IEEE Security & Privacy, Volume 8, Issue 3, pp. 36–44, May–June 2010.

[13] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani Crow droid. Behaviour-Based Malware Detection System for Android. In Proceedings of the Workshop on Security and Privacy in Smartphone's and Mobile Devices (SPSM'11), Chicago, IL, USA, October 17, 2011.

[14] L. Yihe. An Information Security Model Based on Reputation and Integrality of P2P Network. In Proceedings of 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing, Wuhan, Hubei, China, April 25–26, 2009.

[15] L. Qi. Network Security Analysis Based on Reputation Evaluation. In Proceedings of 2011 International Conference on Information Technology, Computer Engineering and Management Sciences (ICM 2011), Nanjing, China, September 24–25, 2011.

[16] http://developer.android.com/reference/android/content/Context.html

[17] http://developer.android.com/reference/android/content/Context.html

[18] Lucas Jordan, Pieter Greyling, "Practical Android Projects" Apress, 2011.

[19] H. Bing. Analysis and Research of Systems Security Based on Android, In Proceedings of 2012 Fifth International Conference on Intelligent Computation Technology and Automation (ICICTA), Zhangjiajie, Hunan, and January 12-14, 2012.

[20] B. Berger, M. Bunke, and K. Sohr, An Android Security Case Study with Bauhaus, in the proceedings of 2011 18th Working Conference on Reverse Engineering (WCRE), Limerick, October 17-20.