ANALYSIS AND RESEARCH OF SYSTEM SECURITY BASED ON ANDROID

ANAMEJE JOSEPHAT AZUKA
Chinweanameje16@gmail.com
Computer Science Department, Federal Polytechnic Oko.

ABSTRACT

Android may be a smart mobile terminal operating platform core on Linux. But thanks to its open-source software and programmable framework character, that leads the Android system susceptible to get virus attacks. This paper has deeply researched from the Linux system security mechanism, Android-specific security mechanisms and other protection mechanisms. And on this basis, Android devices have achieved closely guarded on normal state. So that attackers cannot use the kernel module or core library to get highest access permission and be attacked. Meanwhile, to further strengthen the security of Android devices, it enables them to properly handle the high-risk threat. 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 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; Security system.

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.

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.

STATEMENT OF PROBLEM

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, we believe that in addition to the usability concerns, there is not a clear nderstanding 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.

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.

LITERATURE REVIEW

According to T. Bläsing, et al 2010, Android is a software stack for mobile devices that has an OS, middleware and key applications. Android SDK issued to develop android applications . It uses Java programming language. It is planned to run on differing types of devices (C. Orthacker, et.al 2011). Android platform is based on Linux technology. It is composed of OS, interface and application components. It's issuance breaks the monopoly status of Microsoft windows mobile OS and Nokia's Symbian OS. It allows anyone to develop him own applications. So there's an opportunity that a user is probably going to download and install malicious software's written by software hackers.

ANDROID PLATFORM ARCHITECTURE

Android has built in tools. (A. Shabtai, et al 2010). Android platform composed of Linux kernel, system libraries, android runtime, and application framework the non five parts. Android relies on Linux2.6 version. According to M. Ongtang 2009, It provides core system services security, memory management, process management, network group, driven model. The core part is similar to an abstract level between the hardware layer and other software within the systems.

ANDROID RUNTIME

Android runtime consist soft work components. First,a set of core libraries. Second, the Virtual machine Dalvik. Java programs are received and translated by the VMDalvik. Applications will been capsulated in Dalvik. AVM is available for every and each program vent though some programs are running in parallel.

APPLICATION FRAME WORK

An application frame work is a software framework that's used to implement a typical structure of an application for a selected OS. Any application can publish it's own features. These functions can be used by any other application. (E. Chin, et al, 2003).

Now like most of the main software and operating platforms on the world Android also comes with a software development kit which is termed commonly as Android SDK. Android SDK provides you the AP Ilibraries and tools for building and developing new applications on Android operating environment using the java programming language. According to M. Ongtang, et al 2009, this procedure of developing the applications on Android platform in java programming language using the tools and AP Ilibraries provided by Android SDK is named as Android Application Framework.

BASIC FEATURES SUPPORTED ANDROID APPLICATION FRAMEWORK

In the above mentioned list we did not mention some of the hardware dependant features as the set end to largely vary the device, though nevertheless android application framework support them. Some of the device dependant features supported by android include GSM telephony ,network connection profiles (W. Shin, et al, 2009) such as Bluetooth, Edge, 3G, WiFi, utility features such as camera, compass, GPS, etc.

APPLICATIONS

Applications are written in Java programming language. The Android SDK tools compile the code into an android package, an archive file with apk suffix. The android software platform comes with a set of basic applications. These applications can run simultaneously.

Android initially came into existence with the sure fire concept that developments are given the ability and freedom to make enthralling Mobile applications while taking advantage of everything that the mobile hand set has to offer.

This particular software or Mobile Application is formed to be open source, thereby giving the chance to the developers to introduce and incorporate any technological advancement.

OPEN HAND SET ALLIANCE

Open Hand set Alliance is an amalgamation of Tech Companies with common and particular interest within the mobile user enhancement experience. Companies like Google, HTC, Motorola, Samsung, TelecomItalia, TMobile, LG, Texas Instruments also as Sony Ericsson, Vodafone, Toshiba and Hawaii are Tech giant supported their core abilities and strengths, while keeping and pursuing the characters and goals of every company, their basic Idea of this joining of hands waste feature-rich mobile experience for the end user. This alliance meant the sharing of ideas and innovation, to bring out the seideasin to reality. This provided the millions and millions of Mobile users the experience that they never had.

Like the Apple iphone, Android OS allows third party developers to innovate and build Applications and software for mobile devices. Android is an open, flexible and stable enough to associate itself with new errand newer evolving Technologies. Android's vastrange of easy to use tools and wide selection of libraries provides Mobile Application developers with the means of a tremendous mobile operating software to come up with the foremost efficient and rich Mobile Applications changing the world of many mobile users.

SERVICES

A service is a component that runs within the background to perform long-running operations. For example, a service might play music in the back ground while the user is during a different application, with an activity.

ANDROID SECURITY

a) Android's Five Key Security Features:

1. Security at the OS level through the Linux kernel
2. Mandatory applications and box
3. Secure inter process communication
4. Application signing
5. Application- defined and user- granted permissions

b) Android System Security

In the default settings, no application has permission to perform any operations that might adversely impact other applications, the OS, and data security.

c) Android Security: System- Level Security Features

The Linux kernel provides Android with a group of security measures. It grants the OS a user-based permissions model, process isolation, a secure mechanism for IPC, and the ability to get rid of any unnecessary or potentially insecure parts of the kernel. It further works to stop multiple system users from accessing each other's resources and exhausting them.

Access control systems have existed for a long time. 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 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 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 International Journal of IT, Engineering and Applied Sciences Research (IJIEASR) ISSN: 2319-4413 Volume 2, No. 2, February 2013 i-Xplore International Research Journal Consortium www.irjcjournals.org 50 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.

ANDROID APPLICATION SECURITY FEATURES

This user based protection allows Android to make an "Application Sandbox." Each Android app is assigned a unique user ID, and every runs as a separate process. Therefore, each application is enforced at the method level through the Linux kernel, which doesn't allow applications to interact with each other, and provides the monthly limited access to the Android operating system. This gives the user permission-based access control, and he/she is presented with an inventory of the activities the Android application will perform and what it'll require to try to to them, before the app is even downloaded. The same goes for file system permissions-each application (oruser) has its own files, and unless a developer explicitly exposes files to a different Android application, files created by one application can't be read or altered by another.

- Android Application Security Scans

Keep the following in mind when performing security tests:
•Inbound SMS listeners (command and control)
•Unsafe file creation
•Improper data base storage
•Unsafe use of shared preferences
•Storage of sensitive data on mass storage device
•Content provider SQL injection
•APN or proxy modification

a) Android Security: Geared Towards User-Friendly Security

All of Android's more technical security measures are designed to be simply presented to the user, meaning that they will be easily controlled through the interface. Straight forward methods of improving your Android device's security can include: using a password or pin, setting your phone to lock after a period of inactivity, only enabling wireless connections that you use, and only installing Android apps you trust and have personally vetted.
Google also only allows tested and proven secure Android applications into its market place, meaning that the user has less of an opportunity of putting in a malicious app. Furthermore, the Android security system prompts the user to permit the installation of an application, meaning that it's impossible to remotely install and run an application.

b) Android system security protection
Android system safety inherited the planning of Linux within the design ideology. In practice, each Android application runs in its own process. In the OS, each application runs with a singular system identity. Most of these curity functions are provided by the permission mechanism. Permission are often restricted to particular specific process operations. Android is privilege separated. Data security mainly relies on software signature mechanism. It uses Android Manifest. Xml file. When specified software services recalled, the system first checks this file. To make use of protected features of the device, one must include in Android Manifest. xml, one or more tags declaring the permissions.

ANDROID ANTI THEFT SECURITY
The ultimate security for Android device just in case it's ever lost or stolen. Advantages of this feature are accurate tracking, encoding, Spy camera activation and Device lockdown. It also validates permissions for send SMS messages, hardware controls, take pictures and videos, your location, fine (GPS) location, receive SMS, read SMS or MMS, edit SMS or MMS, full internet access, read contact data and write contact data.

ANDROID ANTI THEFT SECURITY
The ultimate security for Android device just in case it's ever lost or stolen. Advantages of this feature are accurate tracking, encoding, Spy camera activation and Device lockdown. It also validates permissions for send SMS messages, hardware controls, take pictures and videos, your location, fine (GPS) location, receive SMS, read SMS or MMS, edit SMS or MMS, full internet access, read contact data and write contact data.

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. 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. 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. 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 permissionbased 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.

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 behavior. 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. 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.

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 analyzed 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.

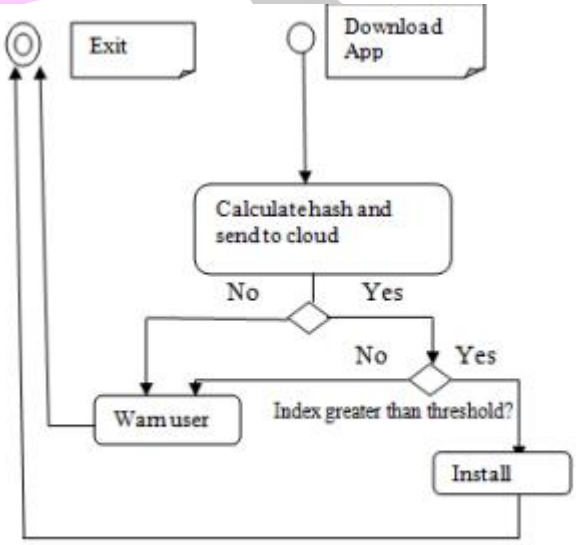

Figure-1: overview of the proposed protocol

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

- 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.
- 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.
- The application has a bad reputation. In this case, the user is warned about the application‘s bad reputation.

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.

Experiment-1

On 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.

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. It also declares the permissions that others are required to have in order to interact with the application's components. 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.

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 helpdevelopers 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.

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 thirdparty 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.

CONCLUSION

We have introduced a methodology to the security community for the empirical analysis of permissionbased 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.

REFERENCES

A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. 2010. Google Android: A Comprehensive Security Assessment. In IEEE Security & Privacy, Volume 8, Issue 2.

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

M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel. 2009. Semantically Rich Application-Centric Security in Android. In Proceedings of the Annual Computer Security Applications Conference (ACSAC '09).

W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka. 2009. 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.

P. Teufl, C. Orthacker, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder and O. Prevenhueber. 2011. 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).

C. Orthacker, P. Teufl, S. Kraxberger, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber.2011. Android Security Permissions.

J. Burns. Developing Secure Mobile Applications for Android 2012. An Introduction to Making Secure Android Applications.

E. Chin, A. Porter Feltm, K. Greenwood, and D. Wagner.2003. Analysing the Inter-application Communication in Android, University of California, Berkeley, Berkeley, CA, USA.