STRATEGIES IN IMPROVING ANDROID SECURITY

Daniel W.K. Tse, Department of Information Systems, City University of Hong Kong, Hong Kong, iswktse@cityu.edu.hk
X. Liu, Department of Information Systems, City University of Hong Kong, Hong Kong, xliu77-c@my.cityu.edu.hk
Christopher Nusaputra, Department of Information Systems, City University of Hong Kong, Hong Kong, cnusaputr2-c@my.cityu.edu.hk
B. Hu, Department of Information Systems, City University of Hong Kong, Hong Kong, biaohu4-c@my.cityu.edu.hk
Y. Wang, Department of Information Systems, City University of Hong Kong, Hong Kong, ywang493-c@my.cityu.edu.hk
M.W. Xing, Department of Information Systems, City University of Hong Kong, Hong Kong, mxing4-c@my.cityu.edu.hk

Abstract

The rise of mobility has witnessed a skyrocket of the number of mobile devices users. Smart phone is almost a must-have for most young adult in today’s society and therefore mobile security is a hot topic issue. The global smartphone shipments have been reported to hit 230 million devices in the second quarter of 2013 with a growth of 47 percent, among which Android operation system captured a record of 80 percent market share. However, along with the popularity, cyber threats increase as well. Due to its high market share and open source architecture, Android has become the most vulnerable mobile operating system to security attacks. Types of security threats such as simple text message Trojans to a more sophisticated rootkits are surfacing every day. In order to seek solutions to Android Security, this paper will begin with the discussion of Android features and characteristics. Then in-depth insights into Android’s system architecture, Play store, and apps etc. will be researched and discussed and finally via analysis this paper will attempt to give feasible solutions to improve Android’s security model from the user’s awareness level as well as technical level.

Keywords: Mobile Security, Android architecture, mobile security awareness, Security Threat solutions
1 INTRODUCTION

With the recent rapid development of mobile technologies, mobile phone has become more and more intelligent and powerful, which is now known as smartphone. Acting as a handheld computer, smartphone possesses its own operating system various applications running on this platform, providing customized services for both individuals and enterprises. Mobile phones also cause security vulnerabilities and threats due to its huge storage of personal information and business privacy. Mobile security has been a great concern in the worldwide. Among the one million mobile devices users, who were reported to be infected by mobile malware in the first semester of 2011 (The Associated Press 2011), only 4 percent use antimalware and antivirus software (Leyden 2011).

Android has become the world most popular mobile operating system due to its features such as open source and easy application development. However, threats also soar in the meanwhile. According to the annual Mobile Threat Report (2013) by Juniper Networks release, Mobile malware grew at a rate of 614 percent during the past year (Kerr 2013), of which 92 percent are targeted specifically at Android. Security of Android OS has raised great concern. Android Security mechanism and multiple security solutions have been designed and tested.

2 LITERATURE REVIEW

2.1 Background Information

Android is an open-source mobile phone operating system based on Linux platform. It is a multi-layer software stack, with each layer providing specific data and services to the above. Android Architecture has four layers: Linux Kernel, Libraries and Runtime, Application Framework and Applications.

Linux Kernel is the foundation of the architecture. Android modified the original code of Linux and made some enhancements to run in an embedded environment. Linux Kernel provides basic functions such as power management, devices driver and networking. The above layer is Libraries, also known as ‘Native layer’. SQLite (database), WebKit (web browser engine), Surface Manager (Display system) and SGL (graphics engine) are included in this layer, which is utilized by upper systems and applications through java interfaces. Android Runtime comprises Dalvik Virtual Machine and Core Libraries. Android applications are written in java but not executed by Java Virtual Machine (JVM). Instead, they run on a custom virtual machine called Dalvik. Compared to the typical JVM, Dalvik combines the Java class files and converts them into Dex format files before application executes on the Android platform. This convert is to decrease the space occupation of .jar files by reuse of the information from multiple class files. Hence, Dalvik enables the light weighed operation system on Android phones. Core libraries in Android provide various functionalities which can be accessed by Java application programming interfaces (APIs) (Dubey & Misra 2013). Next, the Application Framework offers rich varieties of APIs. This is the layer most actively accessed by developers. Components and applications can be accessed and reused by developers under the security mechanism. Last, the Applications include some built-in applications such as browser, calendar and Contact Manager.

Android application is a package of components (Holla & Katti 2012), which are Activities, Service, BroadcastReceiver, and ContentProviders. These four parts interact with each other and each of them can be instantiated (Holla & Katti 2012). Activities act as the visualized user interface of an application, with which users can interact and access the functionality. Usually, an application has multiple activities for navigation. Service enables background running for an application while other application can execute in the front. BroadcastReceiver provides a means to communicate with other applications via Intents. The receiver can be activated by Intent, which is broadcast by applications. ContentProvider is regarded as a repository of data (Holla & Katti 2012). It stores the relevant data and provides applications with the means to share data via Intents. These four components of an application interact together and communication with other applications or components by Intents and various kinds of methods.
Android was designed as an open source platform. To secure a truly open environment, a robust and rigorous security mechanism is definitely required to achieve the security objectives of data confidence, integrity and availability. As a multi-process operating system, Android has developed a professional and multi-layer security program. Security mechanism will be discussed from two levels: Linux mechanism, and Android-specific mechanism (Shabtai, et al. 2012).

Based on the Linux Kernel, Android lays its security foundation on Linux security features. The core objective of Linux Kernel is to isolate the application apks files, so that an application cannot read, write or exhaust other applications’ files and resources. This privilege separation is realized by the mechanism of UID and Sandbox. Access permission is set to limit the access of system files.

Each application is like a user when executing on the system. When installing the application on Android, a unique Linux user ID (UID) is assigned. And Sandbox is created to separate each process. Thus, each application owns a UID and runs on its unique instance of Dalvik Virtual Machine. Applications are limited to interact with each other and access to the operating system by default. But for two applications to share the same process, SharedUserID feature is created. It is required that two applications declare the permission of the use of SharedUserID and possess the same digital signature.

Another mechanism Linux provided is the files access permission check in the kernel level. Apart from the unique user IDs, each file is defined different Read, Write and Execute permissions. As Shabtai, et al. (2012) stated, system files in Android are owned by the ‘System’ or ‘Root’, while application files are owned by a particular application. Linux files access permission mechanism protects files by setting different permissions on files, memories and drivers.

Threats of Android security mainly come from the attacks during data transformation. Malicious applications make unauthorized actions when Android exchanging data through the technology such as Bluetooth, messaging, wireless network and Near Filed Communication. Hence, applications should have limitations when reading or writing confidential data. Mechanism such as application signing, application permissions system and components permissions is established in Android security model.

As no approval processes are set to verify the applications when published by developers on Android Market, applications are required to be signed before publishing. Thus users can distinguish the applications with authentic certificates, which help establish a trust relationship between the users and the developers. Such signing mechanism also supports the SharedUserID and permission mechanisms.

Applications are demanded to declare explicitly of the requirement to access shared resources on the system, or APIs. For developers, they need to determine the permissions the applications required and declare them in the AndroidManifest.xml like this (access to the Internet):

```xml
<uses-permission android:name="android.permission.INTERNET"/>
```

When installing the application, Android system will grant the application permissions by checking the signature and announce the permissions request to users.

When breaking down to the application components level, access permissions are established by the ‘exported’ feature (Shabtai, et al. 2012). When an application set its ‘exported’ feature to ‘false’, the components can be accessed by its own application as well as the application sharing the same UID. Otherwise, when the ‘exported’ feature is ‘true’, then the component can be invoked by other applications, whereas the invoking applications are still subject to the application-level permission mechanism.

2.2 Security Analysis and Solutions

In this part, analysis is performed with regards to the security mechanism discussed above from the perspective of Linux level and application level as well. Moreover, the existed security solutions to the threats will also be discussed.
For the Linux foundation, official Android Security claims that it is a trustworthy platform because as an open source OS, the Linux kernel itself has been researched, tested and debugged by hundreds and thousands of developers. However, according to Shabtai, et.al (2012), Linux kernel is not so satisfactory for its security. They figured out the number of (CVE) Common Vulnerabilities and Exposures that attacked the Linux Kernel was 83 and 73 in the year 2007 and 2008 respectively. Hackers usually attack the drivers by acquiring the ‘root’ access of the system.

As a means of identifying and isolating application resources, sandbox ensures applications to execute in a separate process under its own UID, unable to be accessed by other applications without permission. However, based on the statement by Dubey & Misra (2013), it is inappropriate to depend on sandbox VM to ensure security because there is possibility of breaking out. Also, skeptics of sandbox claimed that more complexity and bugs could be introduced by sandbox. As stated by Chickowski (2012), on the one hand, sandbox is a software platform that is unavoidable to have vulnerabilities that can be exploited. On the other hand, separating all the processes means that users will have to export content out of the sandbox to the underlying device. Such kind of path also contributes to the chance to be exploited (Chickowski 2012). Moreover, in the comparison of Android and iOS security, Chickowski (2012) claimed that iOS has only one sandbox with all the applications running on a single UNIX kernel, which also establishes the limitations for applications to access the files and network. From this perspective, iOS is considered more secure than Android because iOS restricts the users access to application settings.

In order to enhance the Sandbox security mechanism, Bl¨asing et al. (2010) proposed an AASandbox, which is able to perform both static and dynamic analysis to detect malicious actions automatically. This novel design of sandbox aimed at making use of the advantages of both the static and dynamic analysis. Static pre-check is firstly conducted, such as decompilation and decryption. Then a dynamic analysis is performed on the emulator. As a detection system, AASandbox is capable of log the application behaviors from the system level. The resulted log files will be then summarized and reduced to a mathematical vector for better analysis, which facilitates the further development of security mechanism.

Lying in the heart of Android Security mechanism, permission system is positioned to block suspicious malicious actions and enable the normal communication between applications. Pocatilu (2011) pointed out the threats of Android applications due to various sources of application downloading, besides Google Market. Permissions mechanism is inevitably vulnerable to be abused for malware to access the confidential information in phones (Pocatilu 2011). Another vulnerability of permission mechanism raised by Shabtai, et.al (2012) was the SharedUserID feature that allows resource sharing among applications. When two applications are sharing the same User ID, they are both granted the collaboration permission without users’ awareness. Hence, an application is authorized to access an augmented source unrelated to its own functionalities. This creates a chance for communication and information leak between the two applications. Additionally, Dubey & Misra (2013) claimed the un-userfriendly device of the permission mechanism. That is, when installing an application, user is prompted to make decisions on the approval of all the permissions requested or rejection of installation. Such mechanism relies heavily on users’ awareness to secure the phones.

In terms of such vulnerabilities and threats, Shabtai, et.al (2012) suggested the solution of Permission Management Application. It was devised to scan Android application permissions and files access (read/write), so as to detect the installation of suspicious applications as well as undesired permissions. For example, when the scanner detects an unwanted permission, it may either automatically uninstall the application or pop up hints to arouse users’ awareness about the abnormal access of the particular application.

In addition, Powar & Meshram (2013) proposed a policy enforcement Android framework: Android Permission Extension Framework (APEX), which allows users to grant application permissions based on the needs. Check permission methods are created to perform the security permissions checks. After the check, the requested permissions will be evaluated according to the policy repository, which determines whether to grant or deny the permission request. With APEX, users are allowed to specify
the fine grained installation constraints and modifications. However, the policy is not easy to define as the user requirements are to be continuously studied.

Although users are allowed to control the application’s access to the private information, the real usage and data flow by the application cannot be visualized by users. Thus a system-wide information flow tracking tool: TaintDroid, is proposed by Enck, et.al. (2010) to address this problem. TaintDroid provides multiple marking designs to improve efficiency by combing four-layer granularities of information tracking: variable-level, messaging-level, method-level and file-level. With TaintDroid, 30 randomly selected applications are evaluated and the results reveal that two thirds of the applications manipulated data suspiciously. However, there are still some limitations of TaintDroid. One of them is that TaintDroid can only track explicit data flows rather than the implicit flows because tracking control requires static analysis.

The open source feature of Android allows anyone to develop and publish applications into the Android Market. Thus, great numbers of attacks are from the Android Market. Permission-based security model relies on the users’ awareness of malware attack and to block the installation of applications when finding them dangerous. Hence, such security model becomes less reliable on its own. One of the solutions put forward by Abela, et al. (2013) is an automated behavioral analysis system called AMDA. AMDA detects the malware by matching the malware application behaviors with the logs, which are collected by machine learning of the behaviors of know-malware and known-benign applications. It baselines the security state of applications and provides the capability of changes tracking. It can categorize the new uploaded applications on different Android Markets and provide analysis results to users.

Apart from the security mechanisms or frameworks, another important factor influencing Android Security is the users themselves, since the permission decisions depends on the users’ choices. However, the relevant research on user awareness of Android security is relatively limited. Users grant the applications mainly based on the review in the application community. But according to Chia, et.al. (2012), the reviews and ratings in the app markets cannot indicate the privacy risks of an app but rather the functionalities and user experience. Furthermore, a survey conducted by Mylonas, et al (2013) also indicates that users tend to trust the applications download from the app market without awareness of the presence or absence of the test mechanism. Therefore, the user awareness should be considered as a crucial issue in Android Security.

3 RESEARCH METHOD

In order to seek the security solutions and achieve the goal of confidentiality, integrity and availability of Android system, threats of Android security should be identified and assessed. Threats of Android can be divided into two groups: threats caused by attackers and threats caused by user unawareness. Among the threats caused by attackers, malware is one of the main risks because malware can modify the private information in smartphones and abuse the costly services by sending SMS etc. Thus, behavior of the malware will be studied and analyzed, as well as security mechanism of Android.

Comparison is a useful method to find the differences and similarities of different things. We can learn a lot from diversity, diversity is an incredible. In order to show the comparison result clear and well organized, we compare the security performance between Android and iOS. We analyzed security mechanism and vulnerabilities from seven aspects, including system architecture, application distribution, approval process, application permissions, programming language, openness and data protection.

Based on the comparison above, in order to exploit the relationship between user awareness and mobile security, our team conducted online questionnaire survey about with different various level respondents for six weeks. There are 210 records collected; 200 valid records were selected to analyze. Moreover, our respondents contain our peers and our elder relatives coming from various occupations. Based on the above, our questionnaire survey of mobile users has a margin of sampling error of plus or minus 5 percentage points.

For case study, we surveyed three categories of malwares, which have been reported as the top 3 widespread malwares detected on Android mobile phones. According to the 2012 mobile threat report
of Kaspersky Lab, these are SMS Trojan viruses, advertising modules and exploits. During the case study, we found some typical examples and the analysis of its mechanism. We also search for some measures that have been taken to defeat these categories of malwares.

4 RESULTS ANALYSIS

4.1 Comparison Result Analysis

<table>
<thead>
<tr>
<th></th>
<th>Android</th>
<th>iOS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Latest Version</strong></td>
<td>Android 4.4</td>
<td>iOS 7</td>
</tr>
<tr>
<td><strong>System Architecture</strong></td>
<td><strong>Hierarchical Structure</strong></td>
<td><strong>Hierarchical Structure</strong></td>
</tr>
<tr>
<td></td>
<td>1. Applications</td>
<td>1. Core OS layer</td>
</tr>
<tr>
<td></td>
<td>2. Application Framework</td>
<td>2. Core Services layer</td>
</tr>
<tr>
<td></td>
<td>3. Libraries</td>
<td>3. Media layer</td>
</tr>
<tr>
<td></td>
<td>4. Linux kernel</td>
<td>4. Cocoa Touch layer</td>
</tr>
<tr>
<td><strong>Application distribution</strong></td>
<td>Google Play &amp; Other distribution channels</td>
<td>Apple store</td>
</tr>
<tr>
<td><strong>Approval Process</strong></td>
<td>Doesn’t offer certification</td>
<td>Rigorous certification process</td>
</tr>
<tr>
<td><strong>Application Permissions</strong></td>
<td>Users choose for Apps</td>
<td>Same access permission</td>
</tr>
<tr>
<td><strong>Programming Language Used</strong></td>
<td>Java programming language</td>
<td>Objective-C programming language</td>
</tr>
<tr>
<td><strong>Openness</strong></td>
<td>Open source platform</td>
<td>Nearly closed platform</td>
</tr>
<tr>
<td><strong>Data Protection</strong></td>
<td>Access code PIN</td>
<td>Delayed lock code</td>
</tr>
</tbody>
</table>

*Table 1: Comparison of Android and iOS*

For Android OS, any software developer can upload their applications to Google Market. Google will not verify the safety of these applications. However, that does not mean Google stands by and let malware proliferate. Google offers a permission list for users to select the proper access application permission. For instance, it is reasonable for a networking app to request for the Internet access, but not for a simple game app. Additionally, Google will remove the app when there are excessive negative reviews.

For iOS, Apple has its own App store. Every application needs a rigorous certification process when they are uploaded to App store. The approval process is quite complicated, not only will they test the vulnerability, security and maintainability of the app but also inspect the content of the app. Some app involves sex and violence, which is not suitable for children. Apple will remove these kinds of apps to provide a healthy network environment. From this, we can say that Apple has strong sense of safety awareness and responsibility.

As mentioned above, when users install an Android application, the permission list provided by Google enable users to know the actions the application may conduct. But it is worth noting that the application still could carry on some malicious attacks on mobile devices within the range of permission. In fact, there is no concept of “access permission” in Apple’s App. Every application has the same access permission. When the installation is approved, the app will own all access permission without user selection.

Java is Android applications’ programming language. As per OWASP, applications written in interpretation languages like Java are immune to buffer overflow, which thus makes the Android platform somewhat resistant to buffer-overflow attacks. Even if buffer overflow takes place, each application runs in its own Virtual Machine and thus the overflow would neither affect any other application nor would lead to information leakage (unless they are shared applications) (Shetty 2011).

iOS applications are written in Objective-C programming language. Applications are linked to C libraries and vulnerabilities in these standard libraries can cause vulnerabilities even in programs written in "safe" languages. The usage of common C string-handling routines like strcat, strcpy, gets, etc. are predominant in iOS applications and this makes these applications susceptible to buffer-overflow attacks (Shetty 2011).
Android is an open source platform, where all resources are open for testing by users. While this feature makes Android more vulnerable, the speed of malware testing is also increasing. It also accelerates the software upgrading and encourages more innovative people to involve in the application development. Contrast to Android, Apple has a closed source framework. Every application needs a rigorous certification process when they are uploaded to App store. It is safer to download in App store. But it doesn’t mean that there is not malicious software in App store. When a malware pass the censorship and enter the App store, the speed to find and delete the malware is much slower than Android. For Android OS, users can create access password PIN or idle-time screen locking to protect the device itself. Every time you intend to use your phone, you have to enter password to login in. Without a valid password, you can’t access the mobile device. Android does not have delayed Auto-lock feature. It’s an inconvenience for some users, because they have to re-enter code again and again even if they just answered the phone for one minute. A few users even cancel the password to save a lot of trouble. This behavior will enable their phone expose to unsecure environment and their privacy are more easily to leak.

For Apple users, it is relatively convenient to handle the code. Apple has delayed Auto-lock feature. The timeout for system lock is user defined. Users are able to set up the lock time according to their own demands. Besides, Apple can locate your phone when it’s lost. This feature is provided by apple as a feature of its operating system and accompanying online service. If your phone is lost or stolen you can wipe sensitive data from your device. In the event that the phone is returned, you can restore the information from the backups on your desktop. When this feature is enabled, 10 failed passcode attempts will automatically erase data from the device.

From the above comparison, both of Android and iOS have their own strength and weakness of security performance. We cannot make a simple evaluation about which system is better. Actually, we can say both of them are secure but Android offers optional access permission for users to oversee their applications. For the user who has higher secure awareness, Android OS is safety. Otherwise, it will be vulnerability.

4.2 Survey Result Analysis

Regarding the personal users segment, 70% of respondents are using Android mobile when filling in the questionnaire, which indicates a huge portion in mobile market.

22% of respondents choose iPhones because of the good user experience. Actually, what we expected the reason was the high security level of iOS that drives consumer to buy iPhone. The surprising results also reveal the universally low security.

It looks like users with Android mobile have a clear awareness on its insecurity with only 4% of respondents saying that Android OS is high security. Besides, 31% of them consider Android mobile is easier to be attacked and 29% of them regard the poor security problem as one of the weaknesses of Android OS. But they still choose Android mobile for its cheaper price and having access to download software from multi-channel no matter they are security or not.

Accordingly, they do download software from multi-channel not only from Google Play, because there are amazing 40% amount users that never downloads an app from Google Play, and only 6.02% of them always download their apps from Google Play which are more secure.

We can imagine that the Android OS’s security enough but what if the apps are not so security as they may incur threats to users? The secure system has already alerted that this app is risky or is a malware and user should not install it to your phone, otherwise it may steal sensitive information from user and even threatens your mobile system. Only 19% of users answered no, which means that most users will still install a potentially dangerous app even when the system warns them not to.

Based on the analysis, user awareness brings most risk of smartphones. Thus, the enhancement of user’s security awareness is required to improve. Android security, which is also supported by ENSIA report (European Network and Information Security Agency) (Shabtai, et al. 2009). Numbers of users are unaware of the existence of Android private settings to control the data transmission.
Regarding the enterprise employees segment, 63% of enterprise respondents reported that they have suffered from security threats during their organizational business process, but few have specific solutions in place to resolve these problems and protect information assets. There is no wonder that 75% of employee respondents disregard the mobile security as the first priority compared to other assets security. There should be a corresponding policy done by enterprise to help reduce the current mobile device security threats and improve staffs’ sensitive information security recognition.

With regard to the problems above, what can businesses do to better secure Android phones? In addition to what we have mentioned in the user awareness part, there are still some suggestions and advices as follows. Firstly, enterprise should monitor the apps on employee owned devices from the very beginning when distributing access permissions to employees. Moreover, Bit9 recommends blocking rooted or jail broken devices from access corporate networks, because rooting a device can disable built in security protections. Finally, some recommended functions should be added to enterprise mobile device when purchasing these devices from vendor, like whole device encryption for Android, enabling digital screen locking which means a password is compulsory to access a device, using remote wiping in the event that a device containing corporate data goes missing and the inventory system for software and application.

4.3 Case Study Results Analysis

CASE 1: SMS Trojans

SMS Trojans refer to the malware designed to send short message to subscribe a premium rate service without awareness of users (Tropey 2011). Those Trojans disguise themselves in order to get SMS permissions. According to Kaspersky’s statistics, SMS Trojans have occupied 30% of the whole malwares during the third quarter of 2013, increasing 2.3% compared with the second quarter (Chebyshev 2013). Even though it’s lower than the statistics in 2012; the SMS Trojan is still a big threat to Android security. This Trojan always causes a huge financial loss of customers, because if users didn’t check their balance or account statements, it cannot be found (Ramos 2012).

Figure 1 shows the main framework of mostly SMS malwares. Most of SMS malware has structure similar with this model to disguise as a user-friendly SMS application (Hamandi, et al. 2013). This framework involves one activity and three services.

![Figure 1. SMS Malwares Main Framework](Source: http://ieeexplore.ieee.org.ezproxy.cityu.edu.hk/xpls/icp.jsp?arnumber=6550526)

Main activity component provides a user interface to read and send SMS messages. It is the basic component that supports the launch of Listen Service and the Sender Service. Listen Service is to listen for the incoming SMS messages. It is registered as a broadcast receiver. As has been introduced in the literal review, broadcast receiver is used to receive “intent”. It has two types: normal and ordered. The ordered one allows user to set a high priority. And Android doesn’t limit the intent’s priority that app could set after being given the permission. Therefore, each listen service must be the ordered type. It means each application with listen service could interrupt the spread of “intent” and modify “intent” content. Therefore, what if the malware get the permission, broadcast receiver mechanic is useless to provide protection.

The Sender Service component is designed to prevent message being recorded into database. If users lose credit account and want to find some evidences, nothing can be found. In addition, attackers can add some functions to these service components to make it stealthier. The Boot Service is used to make the previously mentioned services. Its function is related to the Listen Service component.

There are still two more things needed to be emphasized. One is that the Listen Service and Sender Service are very sticky, both of them could rerun, even if terminated by user intentionally. The other is that, at the very beginning of the cyber criminal, the application must get SMS permission, which is not shown in the framework but may be the key point of a successful hacking. The way to trick users...
to gain SMS permission decides how big the SMS malware has influence. Here are two examples of existing SMS malwares which using different method to get SMS permission:

1 **SuiConFo.apk**

SuiConFo.apk is a SMS Trojan found by Kaspersky in the November of 2011. This SMS malware pretended to have the function to help users manage their SMS messages. In fact, it didn’t do any function but send malicious messages. This kind of malware is very tricky, because no one would feel doubtful to give SMS permissions to a SMS management application. This application has spread many European countries and Canada at the time it was discovered. When user installs this application and operate it, it will pretend to have some errors. But when users check their mobile account, it’s emptied.

Denis (2011), the Kaspersky Lab Expert, shows a detailed process of the malware did. This application exploits two main malicious classes. One is called “MagicSMSActivity.class”, which is designed to send SMS messages to the server that attacker owns. The other is called “SMSReceiver.class”, which is designed to prevent incoming messages from specific numbers. In addition, this application provided a public method, called getSimCountryIso in the TelephonyManager class, to get the ISO country code of the SIM card in order to achieve cross-country attack. After the definition of country, this application will send message to register a premium rate service and hide the incoming message, which may show the bill or verification of this non-free service. After that, the application also sends one more message to a uniform phone number, as successful verifications.

2 **Fake Famous Application**

Some SMS Trojans just faked themselves as some famous applications in order to be installed by a large number of users (AegisLab 2012). It will ask for SMS permission, and users will be easy to ignore the reminders because of the reputation of the real application. Compared with SuiConFo, this malware get SMS permission from a legitimate channel, which means it’s harder to be detected.

This kind of faking could be a marketing strategy of attackers. This trick will attract a great many users to install the fake application, especially when the attacker announced to provide some new benefits, such as free, or compatible to various platforms. Attackers achieve this fake by repackaging the original application. There are two categories of repackaging applications (Vidas & Christin 2013).

One is called spoofing, which shows even none of the functions that the original application had. It just looks like the existing application, but when user installs it, it cannot do any expected work or even do nothing. For example, Google Play just published a fake version of the Netflix in October 2011 (Android). It announced that it could support all devices while the legitimate one only supported 2.2 and above version of Android system. But if users install it, it will show the same login screen with the original Netflix, and after users input their information, their credentials will be transported to remote servers and the application will uninstall itself.

The other one is grafting, which means attackers just add some malicious procedures into the original application or modify it. It’s easy to accomplish, because the tools to unzip the APK archive and repackage the application is easy to be found and used. For example, apktool can decode applications and rebuild them after modifying some functions only with two commands.

Obviously, SMS Trojans are destructive and lots of work has been done to solve this problem. A lot of work has been done to solve this problem. In fact, the 4.2 version of Android system has provided a new control measure of premium SMS (Whitwam 2013). When application attempts to send SMS to register a charge service, a notification will be provided for users. User will have another chance to refuse it. Bouncer, the malware detection tools provided by official Android Market, also monitors the SMS service in order to reduce the number of SMS Trojans to expose to users (Adware). But, apparently, it’s not effective, because Bouncer was created in 2012, and there are still a lot of new SMS Trojans that have been downloaded from Google Player.
Moreover, there are numbers of anti-malwares combating SMS Trojans, such as lookout, malware bytes etc. One simple way for security is to deny SMS permission of any app, because based on the survey, few benign apps ask for SMS permission. Thus, the simplest but maybe the most effective way to prevent mobile phone from SMS Trojans is to update Android to the newest version and deny SMS permissions of apps.

CASE 2: Advertising modules

Adware refers to kinds of applications which provide advertisements automatically for the purpose of gaining profits for the developer or publisher (Schwartz 2013). In order to find the impact of adware, a team of researchers at Zscaler ThreatLabZ detected 8000 of the most popular apps on Google Play app store. They found that 22% of them included adware, entertainment apps (50%) and personalization apps (42%) (Felt, et al. 2011).

Adware is also a kind of grayware, which means the companies do some stealthy things, for example, collecting user’s searching records, but don’t intend to harm to users (Spring 2011). Because many popular applications are free, the publishers need to provide advertisements to gain profits. Both of users and publishers benefit from this business mechanic. In this situation, Adware is legal. However, some Adwares are phishing malwares. They provided unwanted advertisement for users, even fraud users. In this situation, Adware is a kind of malware. This case study is focused on malicious Adware which do harm to users.

The main purpose of publisher who provided an adware would either advertise his own products or gain an advertising premium. Almost every kinds of malicious technique could be used by advertising. There are some methods that adware uses:

1 Invasive Advertising

This kind of adware doesn’t provide advertisement in itself application, but invade other legitimate applications to show the advertisement (Erturk 2012). It will pop-up advertisement when the user is focusing on other applications. This adware is very smart, because it can exploit the reputation of other applications and the dependency of users on them. Furthermore, it’s also extremely intrusive for users.

2 Phishing Adware

Phishing Adware is not a typical adware. The purpose of it is not to receive advertising revenue or to promote products (Android rooting), but to lure customers to click the advertisement. At the most of time, this attractive advertisement hides a non-relate link in it. When user clicks the advertisement, it may require user its common permission or private information. This case is that a malware disguise itself to be an advertising application.

3 Advertising Click Fraud

Sometimes, the advertising revenue that advertising networks receive is related to the click rate of advertise ment (Spring 2011). This malware can be employed to click fraudulently on the advertisement. Then, if undetected, the click fraud could help advertising networks gains a huge portion of the fraudulent payment.

This solution is similar with the solution to defend Trojans, botnus or virus. Because the harmful Adware is always combined with malicious function, such as exploiting common permission, gaining root control and faking to be non-malicious. Therefore, users could implement some anti-malware applications and pay more attention to the stealthy behaviors of applications, especially during the time of installation.

CASE 3: Exploit—to Gain Root Access to Smartphones

Root exploit (similar to “Jailbreak”) means the process that users exploit system vulnerability to attain root access (Android rooting). For users, if they got the root control, they could use it to extend the function of its mobile phone. For malware publishers, they got the root control; they could design malwares to get users’ private information and credential. Researchers from University of California,
Berkeley, have chosen 6 most popular Android systems from 2010 to 2011 to count the days that root exploits are exposed, and shown that the percent of time with known root exploit are very high. The least one is 74% (Felt, et al. 2011). It means that the root exploit is exposed only one day per five days.

The reason why android has extremely short rooting days is not just technical problem but more incentive promotion. In fact, most of fast rooting didn’t come from attackers or malware publishers, but from non-malicious smartphone tinkerers (Felt, et al. 2011). The main problem of that rooting is that both attackers and users have the incentive to get the root permission. Because carriers and hardware manufacturers always make some limitation to the android OS, users expect to gain the root control to achieve some functions, such as changing other custom operations, removing some pre-install applications and getting complete system backups. Because of the open sourcing of Android system, it has attracted a large number of talent people to defend or attack it. Nowadays, there are many online forums, such as xda and android central, gathering technical experts and Android amateurs to research Android system’s new vulnerabilities. Once they get a root exploit, they will publish the method online with detailed implement method. With their help, a common person who doesn’t have any technical knowledge also can root his or her mobile phone. And malware authors even don’t need to cost anything to get the rooting method. Nowadays, there are still no versions of Android without rooting. It’s impossible to design a system with no vulnerability. In addition, more and more new technology is or will be applied by mobile devices, following new kinds of vulnerability inevitable. Some smart solutions are expected to find to solve this problem.

5 DISCUSSION

5.1 User Awareness Enhancement Solutions

As the analysis from above, user unawareness is one of the major risks of smartphone security. Hence, security education and training programs can be involved as an indispensable complement to technology controls, so as to raise the awareness of employees to take proper security solutions. Particularly, smartphones can be connected to other mobile devices via wireless network, which is often regarded as a means of attack through the modification on the wireless network. Therefore, training programs can be designed to educate the employees not to access unreliable URL via the external network.

In terms of the security solution about how to improve employees’ awareness, what we would like to add is the importance of security policy standards and practices. Since smartphones have been widely utilized in companies to improve the business efficiency, smartphone security becomes crucial and should be involved in the company-scale security strategy. Apart from the security mechanism discussed in the article, in the scale of organizations the security policy should be considered as the basis for all the following practices, which helps highlight the smartphone security to a strategic level.

5.2 Security Mechanism Improvement Solution

5.2.1 Permission Mechanism

After the comparison of Android and IOS, we found that users with Android system should be smarter than users with ISO system. The security mechanics of Android application markets relied on its users to find stealthy malwares and report them. Even the Official Application Market only provides an automatically scanning mechanic (Bouncer) to help user detect malwares. In addition, compared with IOS which only have one official Application markets, there are so many third-party applications markets and no unified supervision. Most of them allow developers to upload their application without any inspection. Some of them are even the beneficiary of malwares or adwares. Such mechanism is not reliable. As a service industry, Android is responsible for providing the best services for users. It should not rely so heavily on users to solve the problems. In addition, the backgrounds of users are wide, and most of them lack professional knowledge, and the developers of malwares are always expertise of information technology. Even though users have high awareness of
Android security, they also have difficulty to defeat with the expertise all by themselves. Therefore, we attempt to propose two solutions to help users protect themselves.

1. **Selective App Permission Controls**

Android’s most attractive points are freedom of choice and controls over your phone. Users should be able to customize their phone to tailor their needs. But until now, Android is currently unable to/do not give control to the user regarding about security control. In contrary, iOS will ask the user explicitly for features that an app wants to access such as location, contacts. It works flawlessly, and the app still works even though full permission is not given. Granted not every access can be controlled in iOS, it is still more control than what Android gives. When an app is installed from Google Play, a pop up will show the summary of permissions that are required by the app. There are not so many things you could do except accepting everything that has been offered. A better cautionary message would be a special warning popup for sensitive application permissions. The solution we proposed is selective android permissions, which was actually a secret feature in android 4.3 (Frem 2013). Unfortunately it seems that the feature is far from ready since it has been removed from new android 4.4. Another thing that could be integrated to the android stock OS regarding app permission is the detailed information and warning, one by one breakdown of the permission. This is a feature that can be found from some custom ROM (custom android OS) such as Cyanogenmod.

2. **Point System and Technical User Review**

Users’ feedbacks are very valuable information for everyone. Google is using it to track applications, for developers it can be used to improve their application and in a survey done by us, 70% of users stated that they use app reviews to assess the quality of the app. In this discussion, we want to propose a point system for every useful user review and a special section for technical review. A useful review can be easily classified as having 75% or more people agree that your review is useful. Special section for technical reviews will allow only users with developer’s account meaning they have the tools to monitor the app they are review. They can see what the data the app is using or what data the app is sending, etc. Our suggestion can be backed by a recent controversy (Savov 2013), an iMessage app for android that takes your apple email and password through a third party unknown server in China. The app has since then been pulled from the play store and people have been informed to change their passwords. This incident could have cause less damage if there’s a dedicated review section where user can read technical review before installing the app. The point system will act as an incentive for a more well thought out and in depth review. A good review takes time and reviews could be rewarded with gift cards or Google play credit.

<table>
<thead>
<tr>
<th>User Review</th>
<th>Technical Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>By: Average user ★</td>
<td>By: A seasoned Developer ★</td>
</tr>
<tr>
<td>Review content</td>
<td>Technical Data Screenshot/report</td>
</tr>
<tr>
<td>34 of 35 users find this review useful</td>
<td>99 of 100 users find this review useful</td>
</tr>
</tbody>
</table>

Table 2: A mockup of the Review System

5.2.2 **Root Exploit Mechanism**

Rooting is always the first step for attackers. When they got the root permission, they could do a lot of harmful works, such as information thefts and DDoS attacks. Nowadays, mobile devices are gradually replacing PC to achieve many important, such as payment, website browsing and files transmit. This change is good for both vendors and users. The key issue of the rooting problem is that both users and attackers have the incentive to root Android systems. The expected security situation is shown in Figure 2, but the realistic situation is Figure 3. To solve this problem, we suggest two solutions.
1. Root Access Authorization

This solution is expected to change users’ desire of rooting exploits. For Android system designer, it is extremely hard and expensive to design an Android operation system which could defend the attack from both users and attackers. In other choice, vendors could allow users to access root. In most of situation, amateurs who attempt to find a root exploit only want to show off their talents. If vendors provide an official channel for users to gain root permission, the incentive will disappear. In fact, there are many vendors, such as HTC, Asus, Google and Sony allow users to unlock their devices or change their operating system entirely. And also, many useful functions now could be implemented without destroying the original root documents (Android rooting).

2. Rooting and Custom ROM

Rooting is the process of unlocking full access to your phone’s system files, which will result in even more control over your phone. You can alter the Linux system to perform better or add binaries but the main purpose for rooting is to install a custom ROM. ROM comes from the term Read Only Memory and it is where you would install the custom ROM therefore the term custom ROM is used by the modding community. Custom ROM is a custom operating, community made, and it’s based on Android OS. The purpose is to improve performance and experience of users’ android device. The process of rooting your phone and installing a custom rom is not a simple task and is seen as a dangerous activity to the average android user and a big percentage of them do not realize that it exists. While the risk of problem such as bricking your phone is there, the chance is very low and the process is reversible. Despite the misconceptions, the custom rom community is healthy and growing ever since. Cynogenmod, the most popular custom ROM for example in September 2013 raised 7 million USD (Amadeo 2013) in funding and is on their way to become the third largest mobile operating system below iOS after (Soper 2013) their revolutionary easy to use installer goes online in November 2013. Our team proposes for android users to look at the option of installing cyanogenmod or other custom rom because most custom rom patches security holes faster than older android devices especially benefits from this because they are the most vulnerable due delayed upgrades or no longer supported by their carriers.

6 CONCLUSION

This research has two aspects: In the technical part, we explored Android OS secure mechanism, permission, package in depth, while at the same time found several security flaws and proposed some solutions to improve the Android OS safety performance. In the non-technical part, we applied three useful methods - comparison, questionnaire and case study, which could indicate the connection of user awareness and mobile security. By utilizing the comparison between Android and iOS, we found that for users who have higher safety consciousness, no matter what operating system they use, the mobile security always performs better relatively. In case study, we deeply analyzed the attack principles of malicious software for seeking more significant solutions for operating security. Finally, Android solutions are proposed. From user awareness enhancement view, security politics and user training are significant to raise people’s security attention. These solutions can be the complement of Android technical control. Meanwhile, from the view of Android security mechanism, permissions and root access approval control should be better managed.

Android security keeps improving and it is better than ever each day but there are still a lot of things to improve on. Data and privacy have been a hot button issue and especially right now after the recent NSA leaks. With the growing trends, people should start taking mobile security more seriously. Understanding mobile security and how you can protect yourself should be one of everyone’s priorities.
7 REFERENCES


