Android Keylogging Threat

Fadi Mohsen, Mohammed Shehab
Department of Software and Information Systems
University of North Carolina at Charlotte
Charlotte, NC, USA
{fmohsen, mshehab}@uncc.edu

***Abstract*—The openness of Android platform has attracted users, developers and attackers. Android offers bunch of capabilities and flexibilities, for instance, developers can write their own keyboard service-similar to Android soft keyboards-using the *KeyboardView* class. This class is available since api level 3.0 and can be part of the layout of an activity. Users prefer to download and install third-party keyboards that offer better experience and capabilities. However, there are security risks related to users installing and using these custom keyboards. Attackers can build or take advantage of existing third-party keyboards to create keyloggers to spy on smartphones users. Third-party keyboard once activated would substitute the Android standard keyboard, so all keys events pass this app. As results, many attacks can be launched identified by the permissions granted to these apps. The objective of this paper is to present these attacks, analyze their causes, and provide possible solutions.**

***Index Terms*—mobile security, mobile apps, keyboard logging**

I. Introduction

Mobile devices are becoming more popular than televisions globally. They are running by different mobile operating systems (OS) such as Google's Android, Apple's iOS, Nokia's Symbian, Blackberry Ltd's BlackBerry OS, Samsung's Bada, Microsoft's Windows Phone, etc. Mobile operating systems can also be extended by installing different kind of mobile applications (apps). Developing mobile apps is becoming more accessible to developers, and plenty of development resources and support are available. As a result, thousands of applications are now available in the market, some of them are free, others are not. For example, in the first quarter of 2013, Apple customers have downloaded more than 40 billion apps, the developers have created more than 775,000 apps, and generating billions in revenue. Android US market share dips slightly, remains on top as of April 2013 [23]. In a race to one million apps between Google's Android operating system and Apple's iOS, Google declared victory on July 2013 announcing that the Google Play store had more than one million apps in its app store. Android's noticeable popularity among consumers and developers alike is tightly related to its openness and powerful development framework.

The Android's platform openness has triggered a great rise in privacy concerns and malware. For instance, the majority of Android applications require permissions to access the phone resources: phone book, camera, sensors, etc. Therefore, many concerns have risen regarding data privacy [19], [12], [17], [11], [18]. Many solutions were implemented as well [13], [10], [25], [27] to detect and prevent from information leakage. In the other hand, Android Play store has attracted hackers to spread their malicious apps (malware). The most common Android malwares are spyware and (SMS) trojans that: collect private information, send SMSs to premium numbers, record voice calls, etc. Malware is a general term used to refer to a variety of forms of hostile or intrusive software, including: computer viruses, worms, trojan horses, rootkits, keyloggers, and other malicious software. the work of [29] emphasized the importance of understanding these malwares to build effective defense mechanism.

In this paper, we study the possibility of keylogging attacks on Android. Which is a security threat pertaining to Android openness on specific feature, the keyboard. The Android system shows an on-screen keyboard–known as a soft input method– when any text field within the system or application gets focus. Before the soft keyboard, Android supported the hardware keyboard. The keyboards since then have evolved and users have learned to customize their keyboards and install new ones. The need for customized keyboard varies, multi-language support, different themes, or people with disabilities or health problems [26]. Android provides accessibility features and services for helping these users navigate their devices more easily. Android provides an extensible input method framework that allows applications to provide users alternative input methods, such as on-screen keyboards or even speech input. Android developers use *KeyboardView* to build their keyboards to work within their applications' context (embedded) or as a stand-alone application. In this paper, we focus only on stand alone third-party keyboards. This facility poses some security threats on phone users' data. Given that most data passes through keyboards, users' privacy can be greatly impacted with malicious keyboards.

In this paper, we show the potential risks related to downloading and installing third-party keyboards. The risks are centralized around spying on whatever users enter via like these keyboards. Moreover, the attacks are dependent on the permissions requested by these applications upon installation. Our contributions are as follows:

1) Conducted an empirical study on a set of keyboards collected from the market
2) Provide different scenarios for possible attacks and implementing them

COLLABORATECOM 2013, October 20-23, Austin, United States

DOI 10.4108/icst.collaboratecom.2013.254209

3) From the empirical study and previous work we identify dangerous permissions related to the attack scenarios
4) Develop an Android application, KBsCheckers, detects all running keyboard services, identify potential risks and alert the user to take some actions.

II. Background

In this section, we give background information on Android keyboards, Android permission system, and the key-logging threat.

A. The Evolution of Android Keyboards

The Android operating system started on October 22nd, 2008. It was designed primarily for touchscreen mobile devices such as smarphones and tablet computers. The initial release was missing many features that we consider nowadays necessities, for instance, the on-screen keyboards. The evolution of Android keyboards has gone through many stages:

1) **Android 1.5** Known by its codename, Cupcake [3], supported both virtual and physical keyboards. The virtual keyboard supports both landscape and portrait orientation modes and works with the built in applications (e.g. SMS, web browser, etc.) and third-party applications. It also provides auto-correct capability, a suggestion algorithm and dictionary of suggestions, and support for custom user dictionaries. Moreover, it supports tactile feedback using screen vibration. Finally, it integrated the hooks necessary for third-party developers to develop their own customized keyboards.
2) **Android 2.0/2.1** Known by its codename, Eclair [4], introduced some improvements over the soft keyboard. For example, Eclair used multitouch data on the keyboard to detect secondary presses while typing rapidly, it resulted in improving the accuracy especially for fast typists.
3) **Android 2.3** Known by its codename, Gingerbread [5], improved the keyboard design and functionality. The design and coloration of the keys changed significantly. The multitouch on the other hand has also improved with "chording", allowing users to press multi-key combinations to quickly access the secondary symbol keyboard.
4) **Android 4.0** Known by its codename, Ice Cream Sandwich [6], the changes were made to improve the correction intelligence with an attractive implementation of inline spellcheck and replacement.

From the above items, we conclude that Android supported the third-party developers in developing their own keyboards since version 1.5. Moreover, the correction intelligence was the main functionality that received the most attention since then. In other words, Android developers had the motive and the flexibility to develop their own customized keyboards.

B. Android Permissions

Android permission system mandates applications to possess permissions in order to make API system calls. The APIs provide access to system and user resources such that contacts, messages and camera. The permissions are granted by the user upon installation. Application developers declare the required permissions in the AndroidManifest.xml file using the uses-permission tag. For example, an application needs to request the READ_CONTACTS permission to read the user's address book. Once installed, an application's permission can't be changed.

There are normal permissions and dangerous permissions. The former type has lower-risk and gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application upon installation without user consent. The dangerous permission though has higher-risk that gives requesting applications access to private user data or control over the device which could impact the user negatively. Because of that, the system displays these permissions to the user upon installation. The user can then approve or deny them.

Many security threats were connected to the unsafe usage of Android permissions. For instance, applications can be granted more permissions than they actually need, what researchers called a "overprivilege" [16] or "permission gap" [10]. Malware can leverage the unused permissions for malicious purposes, for instance using code injection. The threat level is not only connected to the meaning of a single permission, instead the permissions combinations play an important role in understanding the potential implications. Android keyboard applications can also request to have permissions upon installation, some of these permissions may impose privacy threat.

C. Key-Logging Threat

Keyloggers or also known for keystroke recorders are softwares whose main purpose is to monitor user's keyboard actions. keyloggers in the computer community can be mainly classified into two categories: hardware and software. Hardware keyloggers take the form of small electronic devices used for capturing the data in between a keyboard and I/O port [15]. They have their own built-in memory, the place where the captured data is stored. It can be either plugged into the end of the keyboard cable or installed inside the computer case, or inside the keyboard itself. This type of keylogger is hard to be detected by the anti-viral software or scanners because: it doesn't use any computer resource, it doesn't use computer hard disk for storing keystroke logs, and it can be placed in different locations. Though, the main disadvantage of this type is that they require physical installation. Software keyloggers on the other hand collect keystroke data within the target machine, store them temporary on local storage before sending them to the attacker who installed the keylogger. It could also be the case where keystroke data sent directly without temporal local storage. The Monitoring methods for software keyloggers are operating-system specific. Keyloggers pose security and privacy risks on users. In Android, software keyloggers are the only type that may exist. In that case, the keylogger must have certain permissions to record, store, and send the keystroke data.

III. Android Input Methods

There are two parts in developing a keyboard in Android: the interface and the implementation. We first talk about Android input method framework then we explain how to create new input method.

A. Input Method Framework Architecture

Android input method framework architecture (IMF) is composed mainly of three components: input method manager [8], input method (IME)[7], and client applications. The *InputMethodManager* is the key component that mediates interaction between the other parts. It can be expressed as the client-side API that exists in each application and talks with a global system service which manages the interaction across all processes. Android *InputMethod* interface represents any method that can generate key events and texts, such as text messages, emails, different languages characters, while handling various input events, and send the text back to the application that requests text input. An Android application that contains an instance of *EditText* or *TextView* need not to worry about implementing the *InputMethod* interface, instead it relies on the standard interaction provided by these two components.

Implementing an input method in Android is done through deriving a class from *InputMethodService* or any of its subclasses. It involves providing two types of interfaces: top-level interface and session interface. The former provides full access to the input method and it is only accessible by the system. To ensure that only the system can bind to it, an input method must require that clients should hold the BIND_INPUT_METHOD, otherwise the system won't bind and will consider that method as compromised. The session interface is what client applications use to communicate with the input method.

B. Creating an Input Method

In this section, we show how to create a keyboard in Android, which is an example of input method (IME). Through out the rest of the paper we use the terms keyboard and IME interchangeably.

1) Declaring IME Components in the Manifest: In the Android system, an IME is an application that contains a special IME service. The manifest should contain service declaration, permission request, metadata, intent filter, and an optional "settings" activity. The intent filter must match the action "action.view.InputMethod", the metadata defines the characteristics of the IME service, and the settings activity is to allow the user pass new options.

2) Designing the Input Method UI: Android keyboard (IME) main components are: the layout component, manifest entry, xml file, and program component. First of all, the *KeyboardView* [2] which is a view that renders a virtual Keyboard. It handles rendering of keys and detecting key presses and touch movements. Like any other Views, *KeyaboardView* should be included in the layout file of the application. Second, the *Keyboard* class [1] loads an XML description of a keyboard and stores the attributes of the keys. A keyboard consists of rows of keys. For instance, we could define myKeyboardView as an instance of a custom implementation of *KeyboardView* that renders myKeyboard, which is an instance of *Keyboard* class . There are some UI design considerations for IMEs, like handling multiple screen sizes and handling different input types (e.g. Text, Numbers, URL, etc.). Developers can handle these issues by modifying the XML files and program section.

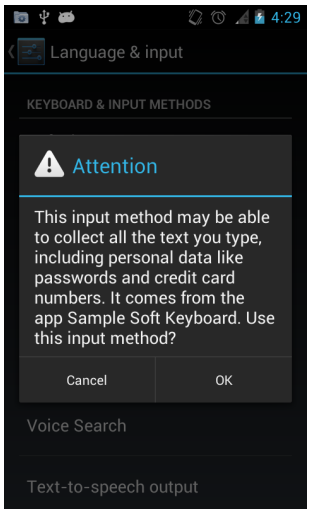

Fig. 1. Warning message upon activating the new keyboard

3) Sending Text to the Application: The purpose of the IME (e.g. keyboard) is to provide the interface, handle user events and then send the text to applications. As the user inputs text with the active IME, text can be sent to the application by sending individual key events or by editing the text around the cursor in the application's text field. In both cases, an instance of *InputConnection* is required to deliver the text. This instance can be retrieved by calling *InputMethodService.getCurrentInputConnection()*.

IV. Threat Model For Mobile Platforms

In this section we present an overview on the security challenges mobile platforms are facing. We present attacker's motives and goals, attack vectors and mobile malwares. The overview will set up the basis before discussing the risks of Android *KeyboardView* or Android third-party keyboards, thus, displaying these risks on regard to the overall risks.

A. Motivation

In smarphones arena, attacker's motives can be classified into two main motives. The first one is to harm the user by disrupting the normal operation of a mobile device. The second is done for financial gain. In addition, there is the spying motive that may overlap with any of the two main motives or may not.

1) Financial Gain: For the financial motive, there are many ways to achieve that. Premium rate number billing is one example. In this scenario, attackers set up and register a premium-rate number that when calling or sending an SMS

to it, the caller is billed a premium rate above the normal cost of an SMS or phone call. The revenue is then shared by the attacker, carrier, and the SMS aggregator. Android applications can request permissions to send SMS messages at installation time. The SMS messages can be sent without the user confirmation. Moreover, Android developers usually benefit financially, either by selling their apps or embedding one of the many ad libraries available on smartphone platforms. Some attackers repackage a legitimate ad-supported application then make it available on the market. So, whenever a user installs, use and clicks on the ads the revenue goes to the attacker. More examples like: pay-per-click, pay-per-install, mTAN stealing, etc.

2) Cause Harm: Some attacks are designated to cause harm to the user and the device. Malicious actions that can drain the battery [28], generate huge network traffic, or destroy the data are examples of this type of attack. While these type of attacks have low motive, yet they can cause financial loss.

3) Spying: Attackers write Android applications that allow someone to track and monitor a smartphone user. These applications may collect and export all SMS messages, emails, call logs, GPS locations, or listen to voice messages. Typically, some of these applications are developed by vendors and made available to purchase by an attacker who then gain physical access to the victim's phone. These applications generate revenue for their vendors, from selling them on the Android Market. The motivation for the attacker is then not to achieve revenue but other matters. For example, it can be used by a family (or a business) to monitor people communications without their knowledge. It can also be used by malicious individuals to steal passwords or credit card information.

B. Mobile Malware

Malware, short for malicioius software, is software used or developed by attackers to interrupt computer operation, collect sensitive information, or gain access to private computer systems. It can take the form of scripts, code, and other software. Malware is a general term used to refer to a variety of forms of hostile or intrusive software, including: computer viruses, worms, trojan horses, rootkits, spyware, keyloggers, and other malicious software. Recent studies indicated that mobile malware is on the rise as more businesses and consumers migrate to phones and tablets. Mobile attacks using malware are growing in both numbers and complexity. The work of [29] has identified 1,200 malware samples that cover the majority of existing Android malware families ranging from their debut in August 2010 to October 2011. The study emphasized the importance of understanding these malware as the best mean to defend against them. In this paper, we provide different scenarios for using *KeyboardView* to keylog user inputs.

C. Attack Vector

They are the collective means and gates through which attackers get into the target mobile platforms to achieve specific goals. The goals can be to access system resources (e.g. camera, GPS, microphone, etc.) or stored data (e.g. pictures, emails, contacts, etc.). Mobile platforms contains many attack vectors, which include: Internet, Bluetooth, USB, Mobile Network Services (e.g. SMS and MMS). Attacks would also take advantage of internal vulnerabilities, such that bugs within software, running on the device and processing external data (e.g. WebView and Browser). If the attacker fails to trick the user into installing her malicious software, a physical access to victim's device would be necessary to get that done.

1) Android Keylogging Attack Vector and Adversary Model: A keylogger can be part of a spyware or a legitimate keyboard application. Keyloggers are sometimes part of spyware packages downloaded onto smartphones without the owners' knowledge. Most keyloggers allow keyboard strokes to be captured and stored on the local storage, but some are programmed to automatically transmit data over the network to a remote computer or Web server. An attacker buys a spyware, get physical access to the victim's device and then installs it. An advanced attacker may choose to modify an existing keyboard application available through an official market, to act maliciously. This is possible due to the fact that reverse engineering Android applications is trivial and requires low effort with the assumption that no code obfuscation is used. The modified application can be then made available on (alternative Android Markets) non-market places. In other scenario, an attacker may also choose to develop keyboard application on its own, insert the keylogging code and then upload the final application on the official markets. Morevoer, developers may leverage some existing code, for instance, Google made some sample code available online [21].

D. Android Keylogging Attack

In Android, keylogging [22] attacks is highly possible because since Android 1.5, Android has integrated the hooks necessary for third-party developers to develop their own customized keyboards. A third-party keyboard in this case will reside between the end user and any system or third-party application that requires user input. A similar attacks is man in the middle attack [24]. Thus, the target is mainly the text inserted by the end users using these keyboards. The text can be limited in scope in case of customized keyboard running in an application context and coupled with few view components. It can be tremendous though in case of the keyboards used in substitute of the standard keyboard. In keylogging attacks, the data can be sent directly to the attacker or stored temporarily on a local storage or remote server. Luckily, in Android we can determine the capability of any third-party keyboard or any other application by the permissions it holds. For instance, if a keyboard has the permission android.INTERNET, then that permission will allow it to open network sockets.

In this paper we assume that Keylogging attacks on Android can be done in a two-step or three-step scenario. The three steps are:

1) Collecting: the assumption here is that all data entered by keyboard app is susceptible for being collected and

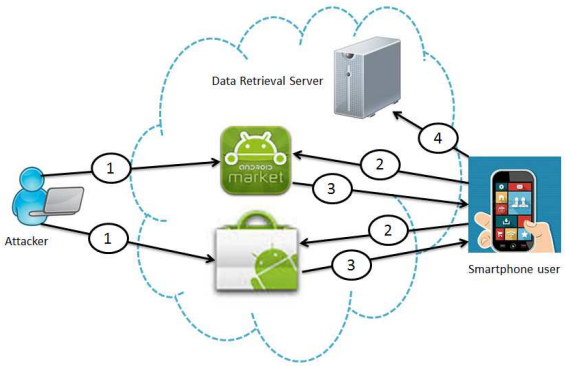

Fig. 2. Android keyboard attack scenarios

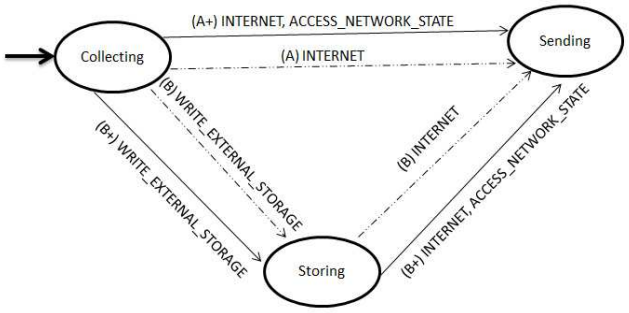

Fig. 3. Android keyboard attack graph

logged. The keyboard can store the input text before sending it to the requested app.

2) Storing: Android provides several options for the developers to save data parmentally. Developers choose the one that fits thier needs, such as whether the data should be private or shared with other applications.
3) Sending: The last step is to send the stolen data to remote places (e.g. servers).

So, in a two-step scenario the attacker writes a code that collects data then sends it to a specific location. In a three-step scenario the attacker writes a code that collects data, stores it temporarily on a local storage, then sends it to a specific location. The user input data could be known to the attacker, e.g. in case of local keyboard coupled with specific text fields. Or, could be any text, e.g. in case of a keyboard app. The processing though is done in case of the data collected by the keyboard app, e.g. pattern matching. We identified the permissions requirements of the two scenario. Kirin [14] used similar approach in that it blocks the installation of apps that request particular permissions combinations. In this paper, the attack model is defined by the requested permissions.

E. Permissions Requirement

The road map to any attack in Android requires that an application possessing some permissions. Starting from the collection step, an application that displays keyboard interface to the end user needs to define a service with the permission, *android.permission.BIND_INPUT_METHOD*. For the storage methods on Android, one can choose from the following list:

1) Shared Preferences: Store private primitive data in key-value pairs.
2) Internal Storage: Store private data on the device memory.
3) External Storage: Store public data on the shared external storage.
4) SQLite Databases: Store structured data in a private database.
5) Network Connection: Store data on the web with your own network server.

The above storage methods aggregately requires the following set of Android permissions: *WRITE_EXTERNAL_STORAGE, INTERNET, ACCESS_NETWORK_STATE*. The last two are also required for sending data over the Internet.

In Figure 3, we distinguish between four possible attack types with third-party keyboards: A, A+, B, and B+. In case of A, a keyboard needs only the INTERNET permission, A+ requires INTERNET and ACCESS_NETWORK_STATE permissions, B is A plus the WRITE_EXTERNAL_STORAGE permission, and B+ is A+ plus the the WRITE_EXTERNAL_STORAGE permissions.

V. Example of a Malicious Keyboard

Users install like these keyboards and let them take over the system for the purpose of enjoying their capabilities and overcoming the system keyboard limitations. They need a keyboard that supports different languages, smarter in terms of auto completion and correction, and offers more symbols. All the data that is entered by the user in any application: system app or third party app, will pass through the new keyboard. In order to demonstrate how user typing could be collected and sent to a remote destination, we present here a simple malicious keyboard application for the Android platform. Android platform adopts permission based security model where user is one of the key security factors. Since the user decide to install certain application and grant it the requested permissions. So, tricking the user to install malicious application is part of the attack model.

The scenario consists of four entities: the attacker, user, application, and Android market. First, the attacker deploys the malicious application into official Android market or alternative market (1). The user using a search engine will look for a keyboard application, finds the malicious on the official market or the alternative (2). She then clicks on the link and installs it on the device after accepting the requested permissions (3). As the user types in using the malicious keyboard, the keystrokes is being sent to a remote server (4).

VI. Data Analysis

We present in this paper the first empirical study on Android third-party keyboards phenomenon, giving many details about these keyboards. In this paper our data analysis is centralized around two major points. The first is concerning the

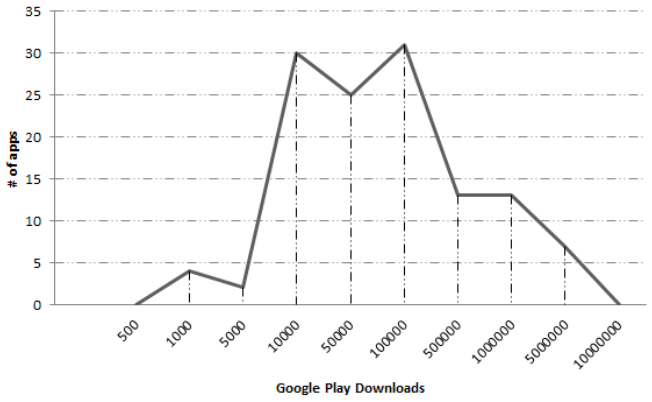

Fig. 4. The download frequency for the collected third-party keyboards

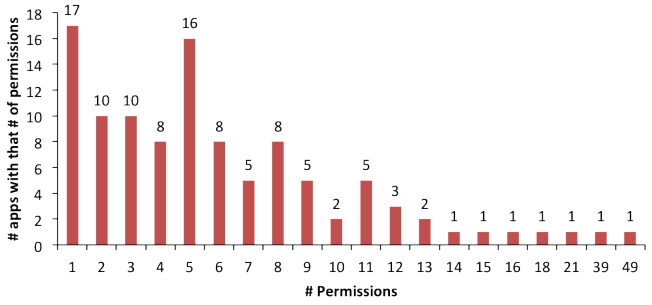

Fig. 5. The frequency for the number of permissions requested by collected apps

number and type of permissions requested by the collected keyboard applications set, which tells us the vulnerability of the keyboard application. The second is the number of users downloading these applications and their ratings, which tells us the scale of an attack. We conducted an empirical study on a set of 125 keyboards apps collected from Google Play Market. We used the apktool to get their *AnadroidManifest* files. We then parse these files for the permission requests. Moreover, we use the data; app installation count and ratings, available on the Android Market information page.

A. Permissions Analysis

Among the studied 125 keyboard applications, only 15% requested zero permissions, the remaining 85% requested one or more permissions upon installation, see Figure 8. The number of permissions requested ranges from 1 to 16 with few exceptions. Figure 7 displays the frequency for each number of permission, 61% of these applications requested more than two and less than eight permissions. The highest frequency is seventeen for one permission. As an extreme case, we found a keyboard application that requested forty nine permissions. The application has a rating of 4.6 out of 5 and number of downloads were between five hundred thousands and one million. For the type of permissions requested, in Figure 6, we show the most requested permissions overall keyboard apps. The permission android.permission.VIBRATE was the

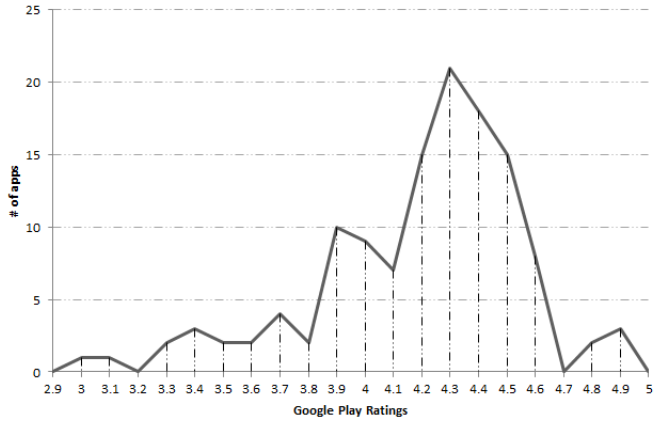

Fig. 7. The rating frequency for the collected third-party keyboards

top most requested permission, 77.60% of the apps requested this permission.

B. Popularity

Number of downloads and users' ratings are two measures for the popularity of any Android application. Thus, as these two measures go higher, attackers get attracted to use like these applications to host their malicious code. Figure 7 shows the ratings distribution among all applications, 62% of the applications have rating range between 4.1 and 4.6. For the number of downloads, in Figure 4 we show the downloads distribution. From the figure, there are thirty applications with the range of fifty thousands and one hundred thousands downloads. More than 50% of applications have download rates between fifty thousands and five millions.

Keyboard apps fall under the "Productivity" category according to Google categorization [20]. AppBrain [9] provides statistics for all Android app categories, follows are some information particularly given for the "Productivity" category:

1) The total number of apps in this category is 21154 apps
2) The average star rating is 4.0 out of 5.0
3) The number of apps in the category that have more than 50,000 downloads is 1700 apps which is about 8% of the apps in the "productivity" category.

Based on our results and AppBrain statistics, we conclude that most third-party keyboards request permissions that would make them vulnerable to some serious attacks: 48.8% of these apps are vulnerable to type A attack, 42% to type A+, 29% to type B attack, and 24% to type B+. The number of downloads as well as users' numerical rating is considerably higher than the average compared to the apps of their category.

VII. Recommendation and Proposed Solution

In this section, we provide a some recommendations and a solution to prevent from spying/keylogging. We also discuss the tool we developed to help users detect potential keylogging/spying applications.

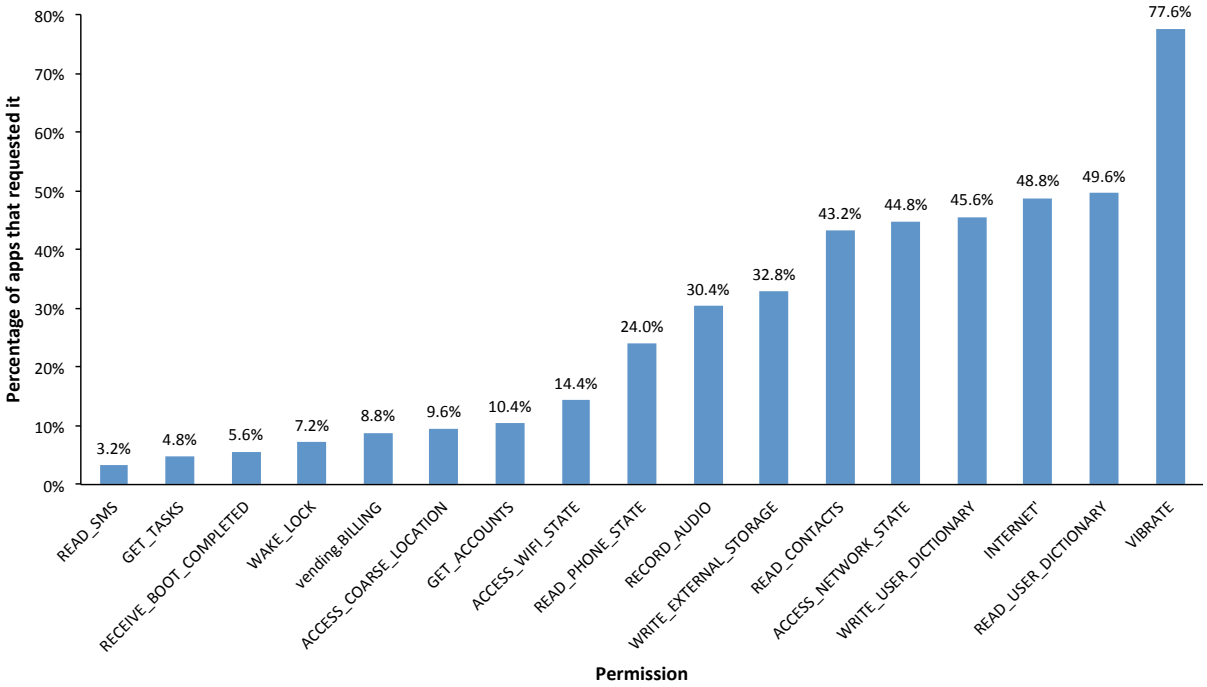

Fig. 6. The 17 most requested permissions and the percentage of third-party keyboards that request them

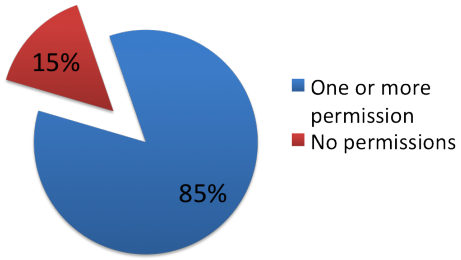

Fig. 8. Among the studied set, 15% requested 0 permission, 85% requested one or more permission

1) Recommendations: In section IV, we describe the attack vector and different attack scenarios. In addition, we provide some recommendations for Android users, to protect them against keyloggers to ensure that they don't become a key logging victim. The following steps would improve the user's chances in subverting a key logging threat:

1) Install applications from the official market especially the keyboard apps.
2) Carefully read the application description, the ratings and users' comments.
3) Upon installation, check the requested permissions thoroughly, look for dangerous combinations.
4) When using a third-party keyboard, be alerted if there is some slowness when typing or if you notice an increase on the usage of Internet data plan.

The Android third-party developers must also be more conservative in the permissions requests they mandate to prevent attackers from using their keyboards to host malicious components. Our analysis showed that the majority of third-party keyboards requires several permissions. Nonetheless, very few require no permissions at all.

A. Proposed Solution

Detecting keyloggers on Android can be done by either monitoring function calls made by the keylogger or by checking its characteristics. Our solution is based on the later method, KBsChecker tool checks for any keylogging indicators and alarms the user if any found. It does some analysis on the installed Android applications specifically reads their permissions and installation information. The tool retrieves these information using Android *PackageManager* and that includes: applications name, package name, version, required features, path info, target SDK versions, installation data, and last modified. Most importantly it retrieves the required permissions for each application and service components.

Now, we scan the services list for each application to determine if it is a third-party keyboard or contains an IME service. In both cases we check the required permissions list and search them for any dangerous combination that pose security threat. At the first page of KBsChecker report is a list of identified third-party keyboards, see Figure 9. The user can click any of them to view its details information and related alert message if they exist. For some attack scenarios the previous information are very helpful, for instance, in case that the attacker gets physical access to the victim device and installs the keylogger app. The current analysis considers malicious keylogging activity to be done by only one installed

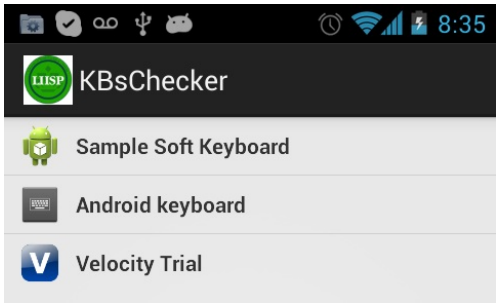

Fig. 9. The first page a user sees when KBsChecker starts.

app. There are other scenarios where more than one app are collaborating to achieve the same goal.

The application requests only one permission and requires low processing and memory resources. The application will be made available in the market for users to install and use.

VIII. Conclusion and Future Work

The paper presented the risks of Android third-party keyboards on user privacy and secrecy. The paper conducted an analysis study on a set of 125 third-party Android keyboards, collected from the market. The number and type of permissions requested by these keyboards poses privacy risks. We defined the attack vector for the possible attacks and the permissions needed to make that happen. The majority of these keyboards requested these permissions, thus posing security risks. We also developed an Android application, KBChecker, that help users find all third party keyboards installed on their device and look for risk indicators. We conclude the work with some advices and recommendations for both users and third-party keyboard developers.

As future work, We will expand our solution to cover scenarios where multiple apps are collaborating to keylog users data. We will extend the current model by adding fourth step, the sharing step. In that case, a malicious keyboard logs user's typing, share it with another malicious app on the system, and that app can then sends it to a remote server. That requires new ways to enumerate new scenarios and their indicators and related permissions. We also planning on implementing and making the solution available on the market.

IX. Acknowledgements

This research was partially supported by grants from the National Science Foundation (NSF-CNS-0831360, NSF-CNS-1117411) and a Google Research Award.

References

[1] Android. Android keyboard class. "http://developer.android.com/reference/android/inputmethodservice/Keyboard.html/", 2013.
[2] Android. Android keyboardview class. "http://developer.android.com/reference/android/inputmethodservice/KeyboardView.html/", 2013.
[3] Android. Cupcake. "http://developer.android.com/about/versions/android-1.5-highlights.html/", 2013.
[4] Android. Eclair. "http://developer.android.com/about/versions/android-2.0-highlights.html/", 2013.
[5] Android. Gingerbread. "http://developer.android.com/about/versions/android-2.3-highlights.html/", 2013.
[6] Android. Icecream. "http://developer.android.com/about/versions/android-4.0-highlights.html", 2013.
[7] Android. Inputmethod. "http://developer.android.com/reference/android/view/inputmethod/InputMethod.html/", 2013.
[8] Android. Inputmethodmanager. "http://developer.android.com/reference/android/view/inputmethod/InputMethodManager.html/", 2013.
[9] AppBrain. Appbrain statistics. "http://www.appbrain.com/", 2013.
[10] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus. Automatically securing permission-based software by reducing the attack surface: an application to android. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 274–277, New York, NY, USA, 2012. ACM.
[11] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in android. In *MobiSys*, pages 239–252, 2011.
[12] E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring user confidence in smartphone security and privacy. In *SOUPS*, page 1, 2012.
[13] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In R. Arpaci-Dusseau and B. Chen, editors, *OSDI 2010, 9th USENIX Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, Oct. 2012. USENIX; ACM SIGOPS, USENIX Association.
[14] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.
[15] Epiphan. Hardware keylogger. "http://www.epiphan.com/", 2013.
[16] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *ACM Conference on Computer and Communications Security*, pages 627–638, 2011.
[17] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: user attention, comprehension, and behavior. In *SOUPS*, page 3, 2012.
[18] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission re-delegation: Attacks and defenses. In *USENIX Security Symposium*, 2011.
[19] M. Frank, B. Dong, A. P. Felt, and D. Song. Mining permission request patterns from android and facebook applications. In *ICDM*, pages 870–875, 2012.
[20] Google. Android categoration. "https://support.google.com/googleplay/android-developer/answer/113475?hl=en/", 2013.
[21] Google. Softkeyboard sample. "http://developer.android.com/tools/samples/index.html/", 2013.
[22] M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. Technical report, 2007.
[23] A. Martonik. Android market share. "http://www.androidcentral.com/android-us-market-share-dips-slightly-remains-top-april-2013/", 2013.
[24] U. Meyer and S. Wetzel. A man-in-the-middle attack on umts. In *Proceedings of the 3rd ACM workshop on Wireless security*, WiSe '04, pages 90–97, New York, NY, USA, 2004. ACM.
[25] M. Nauman, S. Khan, and X. Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.
[26] M. Z. P.A. Condado, R. Godinho and F. Lobo. Easywrite: A touch-based entry methodfor mobile devices. In *13th IFIP TC13 International Conference on Human-Computer Interaction , Workshop on Mobile Accessibility.*, (INTERACT'11), 2011.
[27] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: privilege separation for applications and advertisers in android. In *ASIACCS*, pages 71–72, 2012.
[28] R. Racic. Exploiting mms vulnerabilities to stealthily exhaust mobile phones battery. In *In SecureComm 06*, pages 1–10. SECURECOMM, 2006.
[29] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.