International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 4, Issue 2, February 2015 **ISSN 2319 - 4847**

Effective Risk Detection and Summary Risk Communication for Android Apps

Sachin Bhokare

Amravati University

ABSTRACT

The popularity and advanced functionality of mobile devices has made them attractive targets for malicious and intrusive applications (apps). Although strong security measures are in place for most mobile systems, the area where these systems often fail is the reliance on the user to make decisions that impact the security of a device. Android's current risk communication mechanism relies on users to understand the permissions that an app is requesting and to base the installation decision on the list of permissions. Previous research has shown that this reliance on users is ineffective, as most users do not understand or consider the permission information as it required technical knowledge. Also there is security concern known as Pileup flaws, in which an installed app can get the extra permission without user content while updating the app or OS. I propose a system to provide Summary Risk communication to user in friendly manner which is easy to understand and also provide notifications for Pileup flaws

Keywords: Risk communication, Mobile, Malware, Mobile Security

1. INTRODUCTION

IN recent years smart mobile devices have become pervasive. More than 50 percent of all mobile phones are now smartphones [1], and this statistic does not account for other devices such as tablet computers that are running similar mobile operating systems. According to Google, more than 400 million Android devices were activated in 2012 alone. Android devices have widespread adoption for both personal and business use. From children to the elderly, novices to experts, and in many different cultures around the world, there is a varied user base for mobile devices. The ubiquitous usage of these mobile devices poses new privacy and security threats. Our entire digital lives are often stored on the devices, which contain contact lists, email messages, passwords, and access to files stored locally and in the cloud. Possible access to this personal information by unauthorized parties puts users at risk, and this is not where the risks end. These devices include many sensors and are nearly always with us, providing deep insights into not only our digital lives but also our physical lives. The GPS unit can tell exactly where you are, while the microphone can record audio, and the camera can record images. Additionally, mobile devices are often linked directly to some monetary risks, via SMS messages, phone calls, and data plans, which can impact a user's monthly bill, or increasingly, as a means to authenticate to a bank or directly link to a financial account through a 'digital wallet'.

This access means that any application (or app) that is allowed to run on the devices potentially has the ability to tap into certain aspects of the information. In the benign case the access is performed to provide Useful functionalities, but in other scenarios it may be used to collect a significant amount of personal information and even as a means to have some adverse impact on a user. Furthermore, the line between benign and malicious is often fuzzy, with many apps falling into a gray area where they may be overly invasive but not outright malicious. Compared to desktop and laptop computers, mobile devices have a different paradigm for installing new applications. For computers, a typical user installs relatively few applications, most of which are from reputable vendors, with niche applications increasingly being replaced by web based or cloud services. In contrast, for mobile devices, a person often downloads and uses many apps from multiple unknown vendors, with each app providing some limited functionality. Additionally, all of these unknown vendors typically submit their apps to a single or several app stores where many other apps from other vendors may provide similar functionality. This different paradigm requires a different approach to deal with the risks of mobile devices, and offers distinct opportunities. The present research focuses on the Android platform, because of its openness, its popularity, and the way in which Android handles access to sensitive resources. In Android an app must request a specific permission to be allowed access to a given resource. Android warns the user about permissions that an app requires before it is installed, with the expectation that the user will make an informed decision. The effectiveness of such a defense depends to a large degree on choices made by the users. Indeed whether an app is considered too invasive or not may depend on the user's privacy preference. Therefore, an important aspect of security on mobile devices is to communicate the risk of installing an app to users, and to help them make a good decision about whether to install a given app.

Android's current risk communication mechanism has been shown to be of limited effectiveness. Studies have demonstrated that users tend to ignore the permissions that an app requests [37], [39], [37], [39], and some recent work

has attempted to overcome some of these limitations. Felt et al. [37] proposed several improvements, including: modifying permission category headers, emphasizing risk, reducing the number of permissions, enabling customized permission lists, incorporating user reviews and rethinking the timing of when and how permissions are granted. Lin et al. [39] proposed an approach which incorporates crowd sourced (via Amazon Mechanical Turk) expectations of which permissions are considered reasonable, and presents these expectations on the permission page (e.g., "95 percent of users were surprised this app sent their approximate location to mobile ad provider."). Kelley et al. [37] introduced a concept of "privacy facts" which conveys at a high level the types of information an app has access to (e.g., personal information, contacts, location, etc.), and proposed that these facts be displayed these facts on the app's main description page. I consider an alternative approach to this problem which aims to minimize the space that is used to present information while helping a user make installation decisions with a better understanding of the security and privacy implications

Android is a fast evolving system, with new updates coming out one after another. These updates often completely overhaul a running system, replacing and adding tens of thousands of files across Android's complex architecture, in the presence of critical user data and applications (apps for short). To avoid accidental damages to such data and existing apps, the upgrade process involves complicated program logic, whose security implications, however, are less known. Research made on this brought to light a new type of security-critical vulnerabilities, called Pileup flaws, through which a malicious app can strategically declare a set of privileges and attributes on a low-version operating system (OS) and wait until it is upgraded to escalate its privileges on the new system. Specifically, it is found that by exploiting the Pileup vulnerabilities, the app can not only acquire a set of newly added system and signature permissions but also determine their settings (e.g., protection levels), and it can further substitute for new system apps, contaminate their data (e.g., cache, cookies of Android default browser) to steal sensitive user information or change security configurations, and Prevent installation of critical system services

1.1 Objectives

- To provide solution that leverages a method to assign a risk score to each app before installing it and display a summary of that information to users in friendly manner which is easy to understand. So that user can identify potentially risky apps.
- To provide solution that leverages a method to assign a risk score to each installed app and display a summary of that information to users in friendly manner which is easy to understand. So that user can identify potentially risky apps.
- To provide the solution for Pileup flaws, by sending notification to users whenever the apps gets auto update and gains some extra permission without user consent. This will alert the user about apps abnormal behavior and he can accordingly decide whether to uninstall or keep the app.
- To provide the functionality to uninstall the selected app if user finds it malicious.
- To provide the functionality to block the specific permission of the selected app if user finds it malicious

2. LITERATURE REVIEW/SURVEY

2.1 Security and Usability

Information security and privacy are issues for users of all types of electronic devices. With regard to smart phones, users are more concerned with privacy on their phones than on computers, and they especially worry about the threat of malicious apps [37]. However, although people are shown the permissions an app requests before it is installed, they do not understand them well [37], [39]. Among the recommendations made by Chin et al. [37] was to provide "new security indicators in smartphone application markets to increase user trust in their selection of applications".

The addition of new security indicators not only may decrease the frequency of risky user behaviors, but it may also facilitate the use of smart phones for online transactions by more individuals. Staddon et al. [37] found that user's engagement and perception of privacy are strongly associated, and people spend more time in social networks when they are less concerned about their privacy. This relation may be true as well for app installation. People will not use security features properly if they fail to understand the purpose of the features or the information on which their decisions should be based. The security features also will not be used if the users find the features intrusive or too difficult to master. Therefore, interactions between users and the systems need to be simple and user friendly [38]. Despite this need, studies of various security and privacy measures have shown their usability is typically deficient [38], which often leads to user resistance [39], [37]. Studies have also demonstrated that usability can be improved by systematically studying the human information-processing requirements associated with effective use of the measures and incorporating the resulting knowledge into the designs [37], [37], [38]. Usability of security mechanisms has been studied in contexts other than mobile platforms. Biddle et al. [39] laid out some general ground rules concerning the content of security dialogs; e.g., avoid unfamiliar terms, lengthy messages and misleading or confusing wordings. Schwarz and Morris [39] proposed that web search results be augmented with indicators for helping people assess the degree of trustworthiness of web sources. They found that adding such information to search results is useful, but less

so when the information is added to web pages, presumably because the content, look, and feel of the page dominate the user's judgment. Cranor et al. [39] developed Privacy Bird specifically with the intent to signal to users whether web sites match their privacy preferences. It provides a red bird icon when visiting a site if the privacy policy does not match the user's preferences and a green bird icon if it does match. They extended this idea to web searches with Privacy Finder, which provides similar information when a search engine returns the results of a query [39]. Studies have found the summary privacy information provided by Privacy Bird (and Privacy Finder) to be effective at improving participant's privacy practices [38], [39]. Egelman et al. [37] directly examined the influence of privacy indicators, which showed privacy ratings of online vendors from low to high as one to four green boxes in a row of four that were green), on Internet users' browsing and purchasing decisions. When the privacy indicators were presented alongside the search results, participants who chose to visit only a single website paid more money for a higher level of privacy. However, when this information was provided after a website had been selected, participants did not alter their initial decision to purchase from a cheaper website with lower level of privacy. Finally, Kim et al. [38] proposed the Online Trust Oracle approach for communicating information regarding programs for Windows desktop environment, the interface lists information regarding why a file may be harmful on the left side of the dialogue and why a file may be safe on the right side of the dialog, and it also uses three colors to distinguish programs of different degrees of risk. To summarize, these studies all suggest that presenting high-level summary risk information will be beneficial, particularly if it is displayed early in the selection process.

2.2 Risk Perception and Decision Making

Users make many decisions that affect the overall state of security of any system with which they interact. For security and privacy, most of these decisions relate to the risk to which the individual or system is exposed. Consequently, improving security decisions by users involves taking into consideration factors that influence a user's risk perception and decision making [38]. Also relevant is risk communication,

Which refers to conveying risk to users in a way that allows accurate risk perception and, hopefully, better choices of actions with regard to the actual risks involved [38]. One factor that has been shown to be critical in risky decisions is the way in which losses and gains are framed. With the exact same scenario, the way in which the information is presented can significantly influence the decision-maker's choice. People are risk-averse when the framing highlights positive outcomes, but risk-seeking when it highlights negative outcomes [38], [37].

2.3 Overview of Android Security

The Android system's in-place defense against malware consists of two parts: sandboxing each app, and warning the user about the permissions that the app is requesting. Specifically, each app runs with a separate user ID, as a separate process in a virtual machine of its own, and by default does not have the ability to carry out actions or access resources which might have an adverse effect on the system or on other apps without requesting permission to do so from the user. The permissions consist of capabilities that an app may require such as accessing geo-location information, sending text messages, receiving text messages, and many more. In total there are around 130 unique permissions in Android depending on the version. Each permission has a name, category, and a high level description of what it allows. An example is the "FULL NETWORK ACCESS" permission in the "NETWORK COMMUNICATION" category with its description as "Allows the app to create network sockets and use custom network protocols. The browser and other apps provide means to send data to the internet, so this permission is not required to send data to the internet."

The risk communication mechanism for permissions relies on the assumption that a user understands and makes an informed decision when presented with a list of permissions requested by an app. For most permissions, risks must be inferred because they are not explicitly stated in the description [37]. When browsing a specific app from the Google Play website, a user is able to see details about the app via a series of tabs at the top of the page. In addition to an overview, user reviews, and 'what's new' section, one of these tabs presents the permission information. When an app has been selected for installation, permissions are displayed before the user confirms installation. When app installation is performed directly on the device, there is a Play Store app which allows users to find and install new apps. The options and information are the same as on the website, with the primary difference being that the screen may be smaller and so when information is displayed, including permissions, a user has to make more of an effort to view that information.

2.4 Risk Communication in Android

Studies have shown that Android users tend to ignore the permissions that an app requests [37], [39], [39], and there are many reasons for ignoring them. Permission descriptions are seen as confusing or difficult to understand by many users [39]. Furthermore, nearly all apps request permissions with some associated risk. Felt et al. [39] analyzed 100 paid and 856 free Android apps, and found that "Nearly all applications (93% of free and 82% of paid) ask for at least one 'Dangerous' permission, which indicates that users are accustomed to installing applications with Dangerous permissions. The INTERNET permission is so widely requested that users cannot consider its warning anomalous. Security guidelines or anti-virus programs that warn against installing applications with access to both the Internet and

personal information are likely to fail because almost all applications with personal information also have INTERNET"(p. 6). The implication is that since most apps are considered to be benign, and users see very similar warning information for all apps, the users generally ignore the warnings. Unless a user is highly concerned with security and privacy, and regularly examines the permissions as part of her app selection process, then most likely she has already made the decision to install the app before being presented with the permission information. In Android, a user is able to install the app by clicking a button to 'install' or 'buy' the app. Only then is the user forced to view the permissions that the app is requesting in a final confirmation screen. However, by this point the user has already made the decision to install the app, and this extra warning is often seen as a nuisance and ignored. There is a parallel between Android's permission warning and Windows' User Account Control (UAC). Both are designed to inform the user of some potentially harmful action that may occur. In UAC's case, this happens when a process is trying to elevate its privileges in some way, and in Android's case, this happens when a user is about to install an app that will have all the requested permissions. Recent research suggests the ineffectiveness of UAC in enforcing security. Motiee et al. [39] reported that69 percent of their survey participants ignored the UAC dialog and proceeded directly to use the administrator account. Microsoft itself concedes that about 90 percent of the prompts are answered as "yes", suggesting that "users are responding out of habit due to the large number of prompts rather than focusing on the critical prompts and making confident decisions" [38]. According to Fathi [38], in the first several months after Vista was available for use, people were experiencing a UAC prompt in 50 percent of their "sessions" - a session is everything that happens from logon to logoff or within 24 hours. With Vista SP1, and over time, this number has been reduced to about 30 percent of the sessions. This reduction suggests that UAC has been effective in incentivizing software developers to write programs without elevated privileges unless necessary. The difference between Android and UAC is that UAC encourages the developer to work with fewer privileges since this will lead to a smoother user experience. However, with Android there is no obvious feedback loop to the developer at this point other than the fact that a small fraction of the user reviews may complain about an app being over-privileged.

An effective risk communication approach for Android could provide an incentive for developers to reduce the number of permissions requested by apps, similar to UAC's impact with Windows software developers. By highlighting requested permissions of apps, such risk communication could potentially change user behavior and drive consumption to apps with fewer permissions, thereby creating a feedback loop to developers and having a positive effect on the app ecosystem.

2.5 Privilege Escalation through Mobile OS Updating

Mobile operating systems (OSes) are evolving quickly. Every a few months, major updates or new overhauls of entire systems are made available, bringing to mobile users brand new apps and enriched functionalities. Conventional wisdom is that such a vibrant ecosystem benefits the phone users, making Mobile systems more usable and also more secure, through timely plugging loopholes whenever they are found. Indeed, for years, major smartphone vendors and system/software developers leverage convenient updating mechanisms on phones to push out fixes and enhance existing protection. However, with such updates becoming increasingly frequent (e.g., every 3.4 months for all 19 Android major updates) and complicated (e.g., hundreds of apps being added or replaced each time by hundreds of different Android device vendors), questions arise about their security implications, which have never been studied before. Security hazards that come with software updates have been investigated on desktop OSes. Prior research focuses on either compromises of patches before they are installed on a target system or reverse-engineering of their code to identify vulnerabilities for attacking unpatched systems. The reliability of patch installation process has never been called into question. For a mobile system, this update process tends to be more complex, due to its unique security model that confines individual apps within their sandboxes and the presence of a large amount sensitive user data (e.g., contacts, social relations, financial information, etc.) within those apps' sandboxes. Every a few months, an update is released, which causes replacement and addition of tens of thousands of files on a live system. Each of the new apps being installed needs to be carefully configured to set its attributes within its own sandboxes and its privileges in the system, without accidentally damaging existing apps and the user data they keep. This complicates the program logic for installing such mobile updates, making it susceptible to security-critical flaws. Also adding to this hazard is fragmentation of mobile OSes, particularly Android, and the most popular system. Multiple official Android versions (from Froyo to Jellybean) co-exist in the market, together with thousands more customized by different vendors (Samsung, LG, HTC, etc.). Those versions are slowly but continuously updated to higher ones, leaving the potential adversary a big window to exploit their update installation process, should its security flaws be uncovered. With the importance of this issue, little has been done so far to understand it, not to mention any effort to mitigate the threat it may pose.

3. PROPOSED SYSTEM ANALYSIS/DESIGN

3.1.1 Android System Permissions

Android is a privilege-separated operating system, in which each application runs with a distinct system identity (Linux user ID and group ID). Parts of the system are also separated into distinct identities. Linux thereby isolates applications from each other and from the system.
Additional finer-grained security features are provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform, and per-URI permissions for granting ad hoc access to specific pieces of data.

3.1.2 Android Security Architecture

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another application's files, performing network access, keeping the device awake, and so on. Because each Android application operates in a process sandbox, applications must explicitly share resources and data. They do this by declaring the permissions they need for additional capabilities not provided by the basic sandbox. Applications statically declare the permissions they require, and the Android system prompts the user for consent at the time the application is installed. Android has no mechanism for granting permissions dynamically (at run-time) because it complicates the user experience to the detriment of security.
The application sandbox does not depend on the technology used to build an application. In particular the Dalvik VM is not a security boundary, and any app can run native code. All types of applications — Java, native, and hybrid — are sandboxed in the same way and have the same degree of security from each other.

3.1.3 Application Signing

All APKs (.apk files) must be signed with a certificate whose private key is held by their developer. This certificate identifies the author of the application. The certificate does not need to be signed by a certificate authority; it is perfectly allowable, and typical, for Android applications to use self-signed certificates. The purpose of certificates in Android is to distinguish application authors. This allows the system to grant or deny applications access to signature-level permissions and to grant or deny an application's request to be given the same Linux identity as another application.

3.1.4 User IDs and File Access

At install time, Android gives each package a distinct Linux user ID. The identity remains constant for the duration of the package's life on that device. On a different device, the same package may have a different UID; what matters is that each package has a distinct UID on a given device. Because security enforcement happens at the process level, the code of any two packages cannot normally run in the same process, since they need to run as different Linux users.
You can use the sharedUserId attribute in the AndroidManifest.xml's manifest tag of each package to have them assigned the same user ID. By doing this, for purposes of security the two packages are then treated as being the same application, with the same user ID and file permissions. Note that in order to retain security, only two applications signed with the same signature (and requesting the same sharedUserId) will be given the same user ID. Any data stored by an application will be assigned that application’s user Id, and not normally accessible to other packages using Permissions
A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely impact the user experience or any data on the device. To make use of protected features of the device, you must include in your AndroidManifest.xml one or more <uses-permission> tags declaring the permissions that your application needs.
For example, an application that needs to monitor incoming SMS messages would specify:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.android.app.myapp" >
<uses-permission android:name="android.permission.RECEIVE_SMS" />
   ...
</manifest>
```

At application install time, permissions requested by the application are granted to it by the package installer, based on checks against the signatures of the applications declaring those permissions and/or interaction with the user. No checks with the user are done while an application is running; the app is either granted a particular permission when installed, and can use that feature as desired, or the permission is not granted and any attempt to use the feature fails without prompting the user. Often times a permission failure will result in a SecurityException being thrown back to the application. However, this is not guaranteed to occur everywhere. For example,

the sendBroadcast(Intent) method checks permissions as data is being delivered to each receiver, after the method call has returned, so you will not receive an exception if there are permission failures. In almost all cases, however, a permission failure will be printed to the system log.

However, in a normal user situation (such as when the app is installed from Google Play Store), an app cannot be installed if the user does not grant the app each of the requested permissions. So you generally don't need to worry about runtime failures caused by missing permissions because the mere fact that the app is installed at all means that your app has been granted its desired permissions.

The permissions provided by the Android system can be found at Manifest.permission. Any application may also define and enforce its own permissions, so this is not a comprehensive list of all possible permissions.

A particular permission may be enforced at a number of places during your program's operation:

- At the time of a call into the system, to prevent an application from executing certain functions.
- When starting an activity, to prevent applications from launching activities of other applications.
- Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you.
- When accessing and operating on a content provider.
- Binding to or starting a service.

3.1.5 Android Permission Groups

- In- app purchases

An app can ask you to make purchases inside the app.

- Device & app history

An app can use one or more of the following:

- Read sensitive log data
- Retrieve system internal state
- Read your web bookmarks and history
- Retrieve running apps

- Cellular data settings

An app can use settings that control your mobile data connection and potentially the data you receive**.**

- Identity

An app can use your account and/or profile information on your device.

- Identity access may include the ability to:
- Find accounts on the device
- Read your own contact card (example: name and contact information)
- Modify your own contact card
- Add or remove accounts

- Contacts/Calendar

An app can use your device's contacts and/or calendar information.

- Contacts and calendar access may include the ability to:
- Read your contacts
- Modify your contacts
- Read calendar events plus confidential information
- Add or modify calendar events and send email to guests without owners' knowledge

- Location

An app can use your device's location.

- Location access may include:
- Approximate location (network-based)
- Precise location (GPS and network-based)
- Access extra location provider commands
- GPS access

- SMS

An app can use your device's text messaging (SMS) and/or multimedia media messaging service (MMS). This group may include the ability to use text, picture, or video messages.

Note: Depending on your plan, you may be charged by your carrier for text or multimedia messages. SMS access may include the ability to

- Receive text messages (SMS)
- Read your text messages (SMS or MMS)
- Receive text messages (MMS, like a picture or video message)
- Edit your text messages (SMS or MMS)

- Send SMS messages; this may cost you money
- Receive text messages (WAP)

- Phone

An app can use your phone and/or its call history.

Note: Depending on your plan, you may be charged by your carrier for phone calls.

Phone access may include the ability to:

- Directly call phone numbers; this may cost you money
- Write call log (example: call history)
- Read call log
- Reroute outgoing calls
- Modify phone state
- Make calls without your intervention

- Photos/Media/Files

An app can use files or data stored on your device.

Photos/Media/Files access may include the ability to:

- Read the contents of your USB storage (example: SD card)
- Modify or delete the contents of your USB storage
- Format external storage
- Mount or unmounts external storage

- Camera/Microphone

An app can use your device's camera and/or microphone.

Camera and microphone access may include the ability to:

- Take pictures and videos
- Record audio
- Record video

- Wi-Fi Connection information

An app can access your device's Wi-Fi connection information, like if Wi-Fi is turned on and the name(s) of connected devices.

Wi-Fi connection information access may include the ability to:

- View Wi-Fi connections

- Bluetooth connection information
- Device ID & Call Information

An app can access your device ID(s), phone number, whether you're on the phone, and the number connected by a call.

Device ID & call information may include the ability to:

- Read phone status and identity

3.2 Proposed System Architecture/Design

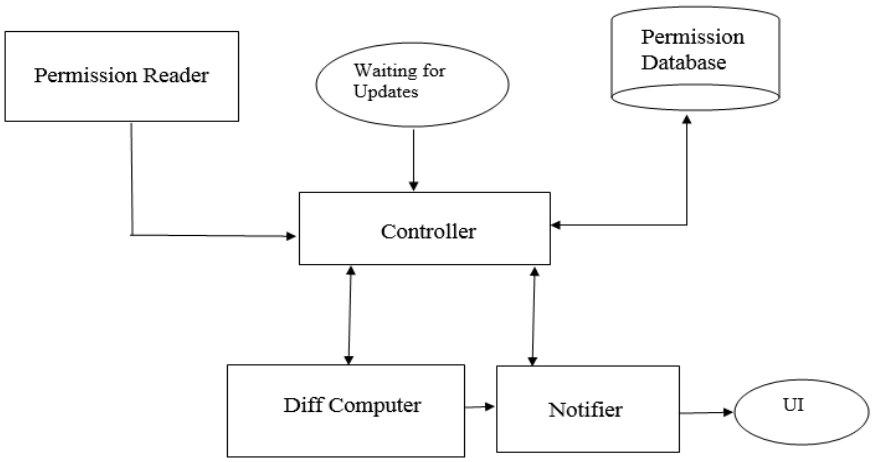

Fig a. Architectural Block diagram of proposed system.

In the above block diagram Permission reader read the permissions required by all apps installed on the device and Controller stores this information in permission database for future reference. Controller also sends this information to Notifier which sends the information in readable format to UI.

Controller keeps on waiting for any apps update, when any app get auto updated, controller gets the notification. On getting notification it reads the permission required by the app/s and send the information to Diff Computer. Diff

computer then compares the info with the information stored in the permission database and if finds any difference in app permission before and after update, it activates the Notifier to send the notification to user regarding the auto permission elevation. User then can make the decision accordingly. If user decides to uninstall the app then controller uninstalls the app from device.

4. CONCLUSION

The expected results from my hypothesis that when risk ranking is presented in a user-friendly fashion, e.g., translated into categorical values and presented early in the selection process, it will lead users to select apps with lower risk. I expect that adding a summary risk metric would cause positive changes in the app ecosystem. When users prefer lower-risk apps, developers will have incentives to better follow the least-privilege principle and request only necessary permissions. It is also possible that the introduction of this risk score will cause more users to pay for low risk apps. Thus, this creates an incentive for developers to create lower risk apps that do not contain invasive ad networks and in general over-request permissions. Also the Pileup flaws can also be reduced.

REFERENCES

[1] Christopher S. Gates, Jing Chen, Ninghui Li, Senior Member, IEEE, and Robert W. Proctor, "Effective Risk Communication for Android Apps" IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 11, NO. 3, MAY-JUNE 2014

[2] Christopher S. Gates, Ninghui Li, Senior Member, IEEE, Hao Peng, Bhaskar Sarma, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, Member, IEEE Computer Society, and Ian Molloy "Generating Summary Risk Scores for Mobile Applications" IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 11, NO. 3, MAY-JUNE 2014

[3] Luyi Xing_, Xiaorui Pan_, Rui Wangy, Kan Yuan_ and XiaoFeng Wang, Indiana University Bloomington "Upgrading Your Android, Elevating My Malware: Privilege Escalation Through Mobile OS Updating"

[4] A.I. Anton, J.B. Earp, Q. He, W. Stufflebeam, D. Bolchini, and C. Jensen, "Financial Privacy Policies and the Need for Standardization," IEEE Security and Privacy, vol. 2, no. 2, pp. 36-45, Mar./Apr. 2004.

[5] D. Balfanz, G. Durfee, D.K. Smetters, and R.E. Grinter, "In Search of Usable Security: Five Lessons from the Field," IEEE Security and Privacy, vol. 2, no. 5, pp. 19-24, Sept./Oct. 2004.

[6] R. Biddle, P.C. van Oorschot, A.S. Patrick, J. Sobey, and T. Whalen, "Browser Interfaces and Extended Validation SSL Certificates: An Empirical Study," Proc. ACM Workshop Cloud Computing Security, pp. 19-30, 2009.

[7] E. Chin, A.P. Felt, V. Sekar, and D. Wagner, "Measuring User Confidence in Smartphone Security and Privacy," Proc. Eighth Symp. Usable Privacy and Security (SOUPS '12), pp. 1-16, 2012.

[8] L.F. Cranor, M. Arjula, and P. Guduru, "Use of a P3P User Agent by Early Adopters," Proc. ACM Workshop Privacy in the Electronic Soc., pp. 1-10, 2002.

[9] L.F. Cranor, P. Guduru, and M. Arjula, "User Interfaces for Privacy Agents," ACM Trans. Computer-Human Interaction (TOCHI '06), vol. 13, no. 2, pp. 135-178, 2006.

[10] N. Dell, V. Vaidyanathan, I. Medhi, E. Cutrell, and W. Thies, "Yours is Better!: Participant Response Bias in HCI," Proc. Conf. Human Factors in Computing Systems, pp. 1321-1330, 2012.

[11] A. Diederich and J.R. Busemeyer, "Judgment and Decision Making," Experimental Psychology, A.F. Healy and R.W. Proctor, eds., second ed., pp. 295-319, John Wiley & Sons, 2013.

[12] S. Egelman, L.F. Cranor, and A. Chowdhury, "An Analysis of P3P -Enabled Web Sites among Top-20 Search Results," Proc. Eighth Int'l Conf. Electronic Commerce, pp. 197-207, 2006.

[13] S. Egelman, J. Tsai, L.F. Cranor, and A. Acquisti, "Timing Is Everything?: The Effects of Timing and Placement of Online Privacy Indicators," Proc. 27th Int'l Conf. Human Factors in Computing Systems, pp. 319-328, 2009.

[14] B. Fathi, Engineering Windows 7 : User Account Control, MSDN blog on User Account Control, http://blogs.msdn.com/b/e7/archive/2008/10/08/user-account-control.aspx, Oct. 2008.

[15] A.P. Felt, K. Greenwood, and D. Wagner, "The Effectiveness of Application Permissions," Proc. Second USENIX Conf. Web Application Development (WebApps '11), 2011.

[16] A.P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," Proc. Eighth Symp. Usable Privacy and Security, 2012.

[17] M.L. Finucane, A. Alhakami, P. Slovic, and S.M. Johnson, "The Affect Heuristic in Judgments of Risks and Benefits," J. Behavioral Decision Making, vol. 13, no. 1, pp. 1-17, 2000.

[18] M. Gondan, C. G€otze, and M.W. Greenlee, "Redundancy Gains in Simple Responses and Go/no-Go Tasks," Attention, Perception, &Psychophysics, vol. 72, no. 6, pp. 1692-1709, 2010.

[19] K.A. Juang, S. Ranganayakulu, and J.S. Greenstein, "Using System-Generated Mnemonics to Improve the Usability and Security of Password Authentication," Proc. Human Factors and Ergonomics Soc. Ann. Meeting, vol. 56, no. 1, pp. 506-510, 2012.

[20] D. Kahneman, Thinking, Fast and Slow. Farrar, Straus and Giroux,2011.

[21] P.G. Kelley, S. Consolvo, L.F. Cranor, J. Jung, N. Sadeh, and D.Wetherall, "A Conundrum of Permissions: Installing Applications on an Android Smartphone," Proc. Workshop Usable Security (USEC '12), Feb. 2012.
[22] P.G. Kelley, L.F. Cranor, and N. Sadeh, "Privacy as Part of the App Decision-Making Process," Proc. Conf. Human Factors in Computing Systems (CHI '13), pp. 3393-3402, 2013.
[23] T. H.-J. Kim, P. Gupta, J. Han, E. Owusu, J. Hong, A. Perrig, and D. Gao, "OTO: Online Trust Oracle for User-Centric Trust Establishment," Proc. ACM Conf. Computer and Comm. Security, pp. 391- 403, 2012.
[24] J. Lin, S. Amini, J.I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang,"Expectation and Purpose: Understanding Users' Mental Modelsof Mobile App Privacy Through Crowdsourcing," Proc. ACM Conf. Ubiquitous Computing (UbiComp '12), pp. 501-510, 2012.
[25] R.D. Luce, Response Times: Their Role in Inferring Elementary Mental Organization. Oxford Univ. Press, 1986.
[26] S. Mishra, M. Gregson, and M.L. Lalumi_ere, "Framing Effects and Risk-Sensitive Decision Making," British J. Psychology, vol. 103, no. 1, pp. 83-97, Feb. 2012.
[27] S. Motiee, K. Hawkey, and K. Beznosov, "Do Windows Users Follow the Principle of Least Privilege?: Investigating User Account Control Practices," Proc. Sixth Symp. Usable Privacy and Security, 2010.
[28] H. Peng, C.S. Gates, B.P. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita- Rotaru, and I. Molloy, "Using Probabilistic Generative Models for Ranking Risks of Android Apps," Proc. ACM Conf. Computer and Comm. Security, pp. 241-252, 2012.
[29] E.E. Schultz, "Web Security, Privacy, and Usability," Handbook of Human Factors in Web Design, K.-P.L. Vu and R.W. Proctor, eds., pp. 663-677, CRC Press, 2011.
[30] J. Schwarz and M. Morris, "Augmenting Web Pages and Search Results to Support Credibility Assessment," Proc. SIGCHI Conf. Human Factors in Computing Systems, pp. 1245-1254, 2011.
[31] J. Staddon, D. Huffaker, L. Brown, and A. Sedley, "Are Privacy Concerns a Turn-Off?: Engagement and Privacy in Social Networks," Proc. Eighth Symp. Usable Privacy and Security (SOUPS '12), pp. 1-13, 2012.
[32] S. Sternberg, "Inferring Mental Operations from Reaction-Time Data: How We Compare Objects," An Invitation to Cognitive Science: Methods, Models and Conceptual Results, D. Scarborough and S. Sternberg, eds., pp. 703-863, MIT Press, 1998.
[33] J. Sun, P. Ahluwalia, and K.S. Koong, "The More Secure the Better? A Study of Information Security Readiness," Industrial Management and Data Systems, vol. 111, no. 4, pp. 570-588, 2011.
[34] A. Tversky and D. Kahneman, "The Framing of Decisions and the Psychology of Choice," Science, vol. 211, no. 4481, pp. 453-458,1981.
[35] W. Van Wassenhove, K. Dressel, A. Perazzini, and G. Ru, "A Comparative Study of Stakeholder Risk Perception and Risk Communication in Europe: A Bovine Spongiform Encephalopathy Case Study," J. Risk Research, vol. 15, no. 6, pp. 565-582, 2012.
[36] K.-P.L. Vu, V. Chambers, B. Creekmur, D. Cho, and R.W. Proctor, "Influence of the Privacy Bird User Agent on User Trust of Different Web Sites," Computers in industry, vol. 61, no. 4, pp. 311-317, 2010.
[37] K.-P.L. Vu, R.W. Proctor, A. Bhargav-Spantzel, B. Tai, J. Cook, and E.Eugene Schultz, "Improving Password Security and Memorability to Protect Personal and Organizational Information," Int'l J. Human-Computer Studies, vol. 65, no. 8, pp. 744-757, 2007.
[38] S. Werner and C. Hoover, "Cognitive Approaches to Password Memorability—The Possible Role of Story-Based Passwords," Proc. Human Factors and Ergonomics Society Ann. Meeting, vol. 56, pp. 1243-1247, 2012.
[39] XF. Xie, M. Wang, R. Zhang, J. Li, and QY. Yu, "The Role of Emotions in Risk Communication," Risk Analysis, vol. 31, no. 3, pp. 450-465, 2011.

AUTHOR

Sachin Bhokare received the B.E degree in Electronics and Telecommunication from Sant Gadgebaba Amravati University and Perusing M.E. degree in Information Technology from Sant Gadgebaba Amravati University.