PermissionWatcher: Creating User Awareness of Application Permissions in Mobile Systems

Eric Struse, Julian Seifert, Sebastian Üllenbeck,
Enrico Rukzio, Christopher Wolf

[1] Horst Görtz Institute, Ruhr University Bochum, Germany
{firstname.lastname}@rub.de
[2] Media Informatics Group, Ulm University, Germany
{firstname.lastname}@uni-ulm.de

Abstract. Permission systems control access of mobile applications to other applications, data, and resources on a smartphone. Both from a technical and a social point of view, they are based on the assumption that users actually *understand* these permissions and hence they can make an informed decision about which permission to grant to which piece of software. Results of a survey conducted for this article seriously challenges this assumption. For instance, over a third of participating Android users were not able to correctly identify the meaning of the permission *Full Internet Access*. We developed *PermissionWatcher*, an Android application which provides users with awareness information about other applications and allows to check on the permission set granted to individual applications. In a field study with 1000+ Android users, we collected data that provides evidence that users are willing to follow security principles if security awareness is created and information is presented in a clear and comprehensive way. Therefore, we argue that it is essential for security policies to take the abilities of the target audience into account.

Keywords: Usable Security, Mobile Phones, Android, Access Rights.

1 Introduction

Modern mobile phones or *smartphones* have become truly ubiquitous computers. For instance, they enable users to edit texts, browse the internet, access all kinds of online services at any place. Also, increasing storage capacities allow users to keep a multitude of data on their devices. Some of these data and files are regarded as highly sensitive and private by the users.

To fully exploit the capabilities of smartphones, modern mobile operating systems allow users to install applications of their choice (also referred to as *app*). Obviously, this creates new threat scenarios: in particular, users unwillingly installing malicious software which steals data or uses the smartphone's resources (*e.g.*making calls, sending text messages). One approach to prevent this is to totally close the system. Users are only allowed to install reviewed and signed

applications. For instance, Apple's *App Store* follows this approach to an extent. A true security review would lead to high costs.

A different approach is leaving the system completely open and allowing users to install whichever application they like. However, this requires rules that regulate what data and which functionalities a certain piece of software may access for security reasons. For instance, in the case of Android, a comprehensive set of *permissions* for applications exist which can be reviewed by a user before installing an application and consequently granting access to the smartphone. However, such a system is entirely based on the assumption that users are familiar with these access rights and are further able to understand them in order to make a qualified decision whether to install an application or not. However, it is not enough that the user understands *one* isolated access right, but many, and in particular their dependencies among each other.

Therefore, this work aims to investigate whether users of smartphones with a permission-based application security model actually do understand these access rights and the implications they have. Further, this work addresses the question if increasing the users' awareness of what a piece of software potentially could do with a specific set of permissions has an impact on their decision which application to install.

Android is a good example for the rule-based application security model and was hence chosen for our investigations. In an initial online study we explored the understanding of Android users of the permission concept. The results show that a large amount of users does not understand basic access rights and what consequences they may result in (cf. sect. 3). Also, a majority of participants indicated that they are willing to uninstall applications which have permission to accessing too many resources of their phones if they were aware of them. Thus, we designed and implemented *PermissionWatcher*, an application for Android phones that analyses permissions of other applications installed on the phone. Based on a custom set of rules, which we developed, PermissionWatcher classifies applications as *suspicious* if any of the rules apply. Through a home screen widget PermissionWatcher increases the user's awareness of potentially harmful applications. In a field trial we collected usage data from 1.000+ different users. About 9% of them used PermissionWatcher to delete suspicious applications directly with PermissionWatcher. For comparison: On 98.7 % of all phones applications with suspicious permissions were found.

The remainder of this article is organized as follows: We discuss related work and illustrate selected basics about Android. In the following, we report on an initial survey in which we assessed the users' understanding of Android permissions. Further, we detail the design and development of PermissionWatcher and further report on a field trial in which PermissionWatcher was tested. Finally, this article draws conclusions and outlines possible next steps.

2 Related Work and Android Basics

Research related to this work can be classified into the following categories: *user interaction*, *smartphone security*, *application market places*, and work concerning the Android system.

User Interaction. Egelman *et al.*[5] shows the necessity to simplify complex security decisions into easy to understand yes/no warnings. In particular, they have investigated phishing sites and used an automated process to rate if a site is dubious or not. If a certain threshold was reached, the site was marked a potentially dangerous, and the user was given the chance to abort loading. Also, there was a potential risk that this engine made mistakes, this was less likely by at least one order of magnitude than the user surfing on a phishing site and becoming a target.

Amer *et al.*[1] consider different ways of displaying warning messages to users. Their main goal it to maximize the impact of a security warning. At the same time, they want to avoid that warnings are perceived as "rude" and also being ignored by users. They show a dramatic difference in user response depending on the way the warning is displayed.

Smartphone Security. General security considerations, but also specific attack vectors for smartphones informed the design of our survey. According to the Microsoft Security Intelligence Report [16] 44.8% of all malware detected, required user interaction for propagation. Note that [16] deals with computers, not smartphones. Still, its information on malware propagation is clearly relevant to this work.

An extensive overview on attack vectors and differences between normal and mobile security can be found in Becher *et al.*[4]. A technology review with a special focus on threats and attacks concerning smartphones is given by Li and Im [15]. The importance to *"scrutinize permission requests*" is highlighted by Hogben and Dekker [14] who analyze risks particularly for smartphone users and give practical recommendations to avoid them.

Market Places and Overall Comparison. Anderson *et al.*[2] examine the application markets for ten platforms, including desktop operating systems, mobile phones, web browsers and social networks. They define *incentives*, *goals* and *stakeholders* concerning application markets, and analyze case studies on each platform. Concerning smartphones, they "*find that these OSes* [Symbian, iOS and Android] *provide significantly more protection to both users and the system than their desktop counterparts, but that there are very important differences among them in terms of a user's control over applications*" (p. 19). Android receives the best protection rating but it is noted that it offers less assurance of system protection compared to the other smartphone operating systems.

Android. To derive specific rule sets (*cf.*Table 1), we considered the following work concerning possible attack vectors in Android.

Shin *et al.*[18] analyze the Android permission-based security system using state machines, leading to a formal security model [19].

There is extensive work on Android malware and data leakage detection by Enck *et al.*. We quote but a few. Enck *et al.*[7] introduce *Kirin*, a framework to enforce security policies in Android. They give a formal representation of the Android security model and develop a policy model featuring a subject-object-rights access matrix.They also present a novel procedure [8] for identifying requirements and creation of rules relating to the analysis of permission sets, and develop a set of rules. Enck *et al.*revise *Kirin* based on these rules and use it to analyze 311 applications. They find five applications with dangerous functionality, and five applications with dangerous but reasonable functionality. Note that our work takes a similar approach on malware detection, however our focus is on awareness and usability; our tool is clearly designed for users who are not knowledgeable in security.

A study on 1,100 applications by Enck *et al.*[6] also detects common misuse of personal information. Enck *et al.*statically analyzed recovered source code of applications, using data flow analysis, structural analysis and semantic analysis. They observe that the International Mobile Equipment Identity is misused as a cookie by many applications, and 51% of (free) applications connect to advertising or analytic networks.

Permissions. The following works focus on the permission system of Android. This is particularly relevant for our work as we focused on permissions both for our survey as for PermissionWatcher.

Barrera *et al.*[3] propose a methodology for empirical analysis of permission-based security. They analyze 1,100 Android applications using self-organizing maps. Felt *et al.*[10] introduce *stowaway*, a tool for detection of *overprivileges* in compiled Android applications. They test 940 applications and argue that one in three applications is overprivileged. Felt *et al.*reason that *"developers attempt to obtain least privilege for their applications but fall short due to API documentation errors and lack of developer understanding"* (Page 11). This is somehow complementary to our approach where we deal with the demand rather than the supply side of Android applications. Another study of Felt *et al.*[11] deals with the effectiveness of permissions. 1,000 chrome extensions (for the Google Chrome browser) and 956 Android applications are analyzed. Felt *et al.*conclude that permissions can be effective, but improvement is possible and necessary. The fact that 93% of the analyzed Android applications have at least one *dangerous* permission is particularly relevant to our work because it implies that users receive warnings about permissions during the installation of almost every application and are hence likely to ignore them.

Furthermore, Felt *et al.*[12] conducted two usability studies (an Internet survey of 308 Android users and a laboratory study of 25 users). They note that only 17% of the participants (in both surveys) paid attention to permissions during application installation, and only 3% of the participants (of the Internet survey) were able to correctly answer all questions on permissions. Again, this

supports our claim that end-users are likely to ignore security warnings if they are presented in an unintelligible way.

2.1 Android Basics

Android is a mobile operating system for mobile devices such as smartphones and tablet computers. It is based on the Linux kernel version 2.6 and was developed by the Open Handset Alliance. Android comprises of an operating system, a middleware and key applications. Source code, along with any data and resource files, are compiled by the Android SDK tools into a single *Android package*. In this context, all code in a single package is considered to be one application.

2.2 Security

Android's security architecture relies on two basic *security mechanisms* [13], namely *sandboxing* and *permissions*. We focus on the latter in this article, but describe both for completeness.

Sandboxing. Each application is given a distinct, constant identity (Linux user ID and group ID). Each application runs in a separate process and can only access files that belong to the same user ID (with an exception concerning data stored on SD-cards). The kernel isolates applications from each other and from the system. This process is assisted by the *Dalvik* virtual machine, which is specifically designed for Android. Still, as any application can run native (C or C++) code, the Dalvik virtual machine is no strict security boundary.

Permissions. In Android, a permission mechanism enforces restrictions on the operations an application can perform. Applications have to statically declare their required permissions at install time to gain access to certain hardware features and user data, and to be able to share resources and data. There is no mechanism to grant or withdraw permissions dynamically. Therefore, it is an *all-or-nothing decision*: either, a user grants all required privileges, or cannot install the application. There is a system of grouping labels into four different categories (normal, dangerous, signature, signatureOrSystem). For more detailed discussion on Android permissions, cf. Felt *et al.*[10] or Enck *et al.*[9].

All in all, this supports the thesis that any application can circumvent Android's permission system by tricking the user into accepting dubious (from a technical point of view) but plausible (from a user's perspective) permissions.

3 User Understanding of Application Permissions

Departing from the assumption that existing means for regulating application permissions in Android do not meet user requirements in terms of clarity, we designed and conducted a survey to investigate the following hypothesis: (H1) User awareness about application access rights in Android is deficient. In particular,

users do not know and understand the Android security concept and corresponding access rights. Further, (H2) awareness concerning potential threats can be supported by providing users with clear and comprehensible information. To substantiate this hypothesis, we state two additional sub- hypotheses: (H2.1) users are willing to restrict access rights of applications and (H2.2) they would delete potentially harmful applications.

3.1 Setting

In order to investigate these questions we designed a questionnaire targeted to Android smartphone users. The survey included 15 questions structured in the sections 1) smartphone usage and experience, 2) Android application access rights, 3) user attitudes towards privacy, and 4) general understanding of IT security aspects. In addition, demographic data was collected.

We promoted the online survey via four email lists about IT security (with ≈3,000 receivers) and an email list of the students of the computer science department at the (blind for review). As an incentive, each participant who completed the questionnaire automatically took part in a lottery where they could win one of three gift vouchers (value: 15, 10, and 5 (blind for review)). The survey is biased in two ways: First, the participants are far more likely college students than the average population. Second, the number of security professionals is also clearly too large. Interestingly, we could not find this bias to be reflected in the results: People with a security background gave similar answers as participants with other backgrounds. Moreover, even if the survey were biased in terms of more security awareness and security friendly behavior, this would only make our findings *worse* for the general population.

3.2 Survey Results

In total, 113 complete answer-sets were collected from participants (89 male). Overall, they were aged between 17 to 50 years (Mdn=25), while 87% were aged between 20 and 30 years. 73 participants (65%) were college students, while the others had highly diverse backgrounds such as psychologists, social workers, or engineers. Concerning the usage duration of their smartphones, 73% of the participants indicated to own and use their devices for at least six months (45% longer than one year).

Understanding of Android's Security Concept. Participants rated their knowledge of the Android application security concept on a five point Likert scale (1=none; 5=very good). On average, participants rated their knowledge to be mediocre (Mdn=3.0). To assess a general understanding of the Android security concept, we asked the participants whether applications released in the Android Market are subject to a security vetting process, which is not the case. 14 participants (12%) assumed that all applications would go through such as process, 29 (26%) were not sure, and 70 (62%) choose the correct answer. Accordingly, more than a third of users are not sure or assume a security mechanism which is

not existing. Further, we asked if users know at which point application access rights are granted. Three possible answers were provided. Here, a large majority of 91 participants (81%) chose the correct answer ("At application installation"), while 12 (11%) picked the neutral answer, and 10 (9%) choose the wrong answer. 63 (56%) participants answered both questions correctly (3 answered both incorrect) while the remaining picked only one correct. This result does not allow for conclusions concerning the overall understanding of the security concept. Yet, it indicates that fundamental security mechanisms are not fully understood. Concerning the understanding of Android application permissions, we found that participants understand partially what access rights mean. For instance, 99 participants (88%) understood correctly what an application with *read contacts* is allowed to do. However, only 71 participants (63%) knew the correct answer to the question what an application with *full internet access* is allowed to do. Even lower was the percentage of correct answers to the question which actions could be performed by an application with the *make phone call* permission: here only 19 participants (17%) gave the correct answer.

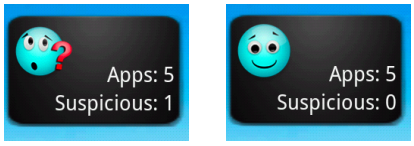
Benefit of Security Mechanisms. Participants were asked to rate their level of agreement to the statement "The available information on Android permissions is sufficient." on a five point Likert scale. A separate neutral answer was available, picked by 10 participants. On average participants rated this statement with 2.0 (Mdn). Further, we asked participants to rate their agreement to the statement "It would be helpful to be able to prohibit access to contacts or sending files to the Internet" where only 3 participants selected the neutral answer. On average, they rated their agreement with 5.0 (Mdn).

In response to the question what they would do if they were warned that an application requires permissions that are potentially harmful, 44% answered that they would delete the application. 28% stated to delete the application in case it would be none of their favorites. 77% of the users agreed to that they would not install the application and search for an alternative. Only 5% stated that they would take no action at all.

In summary, we can conclude that the existing security model is only partially understood by Android smartphone users. This is surprising, as the survey was advertised via email lists received by users with an IT security background. Also, the sample of participants included mainly well educated persons which indicates that even those have difficulties understanding basic Android security mechanism. However, users indicated that warnings concerning security and privacy threats would result in actions such as uninstalling applications which implies that providing users with information would result in a higher level of security.

4 PermissionWatcher Application

In this section, we introduce PermissionWatcher, a mobile application for Android smartphones which increases user awareness about potentially harmful



(a) (b)

Fig. 1. The PermissionWatcher widget: (a) a worried smiley face indicates suspicious applications. (a) after uninstalling suspicious applications, the smiley appears happy.

applications installed on her phone. The concept is based on the assumption that users are willing to take security increasing actions (such as uninstalling a potentially harmful application) once they gain knowledge about a potential threat. In order to increase the user's awareness about potential threats, PermissionWatcher provides a widget that can be installed and displayed on the home screen of the mobile phone. Widgets are a common way to provide users with small pieces of information such as weather information, news tickers, or upcoming assignments taken from the calendar application. Amongst these widgets, the PermissionWatcher widget provides the information of how many applications are installed on the system and how many of these are *suspicious*. When PermissionWatcher detects suspicious applications, a smiley face on the widget emphasizes that the user should take action (see Figure 1). That is, the user can touch the widget to launch the PermissionWatcher application for reviewing details and uninstalling other applications. In case, no applications are detected as suspicious, the smiley face appears as happy.

4.1 Rule Set

PermissionWatcher evaluates the risk of a given application being a potential threat to the user based on a set of rules. As we focus on raising user awareness of Android permissions, this leads to the following two limitations:

1. We inspect single applications. Therefore our rules do not cover permission re-delegation. An application with a certain permission can act as a proxy and perform a task for another application that does not have the permission. Moreover, it is possible to divide a dangerous combination of permissions to separate applications.
2. We do not analyze the code nor the behavior of applications (this would require root privilege or the modification of the Android system). Consequently, we can not detect conventional malware that bypasses the Android security system. For example, we can not detect applications that exercise a root exploit to gain root privileges.

We followed a structured approach to deduce the rule set: First, identifying relevant targets of possible attacks. Then, deriving attack scenarios. This is followed by determining which permission set is required for each scenario which

Table 1. The set of rules applied in PermissionWatcher

Number	Title	Permission Set
1	Relay Contact Data	READ_CONTACTS and INTERNET
2	Relay SMS Messages	READ_SMS and INTERNET
3	Send SMS Messages	SEND_SMS
4	Make Phone Calls	CALL_PHONE
5	Make Phone Calls	CALL_PRIVILEGED
6	Covert Listening Device	RECEIVE_BOOT_COMPLETED, RECORD_AUDIO, and INTERNET
7	Covert Camera Device	RECEIVE_BOOT_COMPLETED, CAMERA, and INTERNET
8	Movement Profile	RECEIVE_BOOT_COMPLETED, INTERNET, and ACCESS_FINE_LOCATION or ACCESS_COARSE_LOCATION
9	Eavesdrop on Phone Calls	RECORD_AUDIO, INTERNET, and READ_PHONE_STATE or PROCESS_OUTGOING_CALLS
10	Relay and Falsify SMS Messages	RECEIVE_SMS, WRITE_SMS, and INTERNET or SEND_SMS
11	Falsify SMS Messages	RECEIVE_SMS and WRITE_SMS
12	Activate Debugging	SET_DEBUG_APP
13	Permanently Disable Device	BRICK

leads to rules based on the permission sets. Finally, similar rules were combined. We have identified three groups of targets:

1. Stored data (*e.g.*contact data, text messages).
2. Hardware features (*e.g.*camera, microphone).
3. Functions (*e.g.*answering calls, send/receive text messages).

We have derived eight different attack scenarios. Examples are *direct monetization*, *attack mTAN based online banking*, or *manipulation of text messages*. For a complete list please refer to sect. 4.2. In the last step, we combined three pairs of rules:

1. Rules that allow creating *movement profiles*.
2. Rules concerning *manipulating text messages*.
3. Rules concerning *eavesdropping calls*.

The resulting rule set includes 13 rules that are used to determine if an application is to be considered as suspicious because it is potentially harmful for the user. The list of rules is given in Table 1. Please note that rules 8, 9, 11, and 12 have been previously defined in [7].

4.2 Scenarios

The following set of attack scenarios were derived:

Extracting information Reading data stored on the device and relay it to an attacker. We focus on attacks that use network connection to relay data. Using text messages to relay data is possible. However, we neglect text messages because they are noticeable by the user (since they may cause costs) and related attacks are complex (data has to be split into separate messages and infrastructure to receive messages is required). Furthermore, we consider the unwanted sending of text messages as a separate attack. We disregard relaying data via *Bluetooth* connections as it depends on physical proximity to the target.

Direct monetization Send *premium rate text messages* or making *premium rate phone calls*. Furthermore, it is possible to use text messages to distribute *junk* messages.

Compromise emergency call system Makeing *emergency calls*. Unwanted emergency calls may cause serious punishment (depending an national laws). In addition, attackers could use phones of numerous targets to attack the emergency call system (a *critical infrastructure*) by constantly making short emergency calls.

Covert surveillance Employing the microphone or the camera to spy on the user. Moreover, it is possible to utilize the GPS sensor to create a *movement profile*.

Eavesdropping on phone calls Use the microphone and system functions to eavesdrop on phone calls. In addition to eavesdropping on conversations, it is possible to extract information from a phone call. For example, Schlegel *et al.*demonstrate how to extract a PIN or credit card number using a *"sound trojan"* [17].

Attacking mTAN based online banking Relay or falsify text messages that contain a mTAN (mobile transaction authentication number). An attacker intercepts a mTAN and uses it to authorize an online banking financial transaction from the victim's account.

Manipulating text messages Falsify or forge text messages. In addition to annoying the victim (for example, by rendering messages useless or displaying *Spam*), the ability to falsify or forge text messages can be a serious security threat. Although falsifying or forging a text message can not be used to relay mTAN, it may enable an attacker to intercept important messages. For example, an online banking system may send alerts or confirmation messages that can be intercepted. Furthermore, forged messages can be used for *phishing* attacks to trick the user into relaying the mTAN.

Dangerous system functions Android offers a function called *brick* to permanently disable the device. According to a related work by Enck *et al.*, it is possible to gain the permission *set debug app* by manipulating the Android API [8].

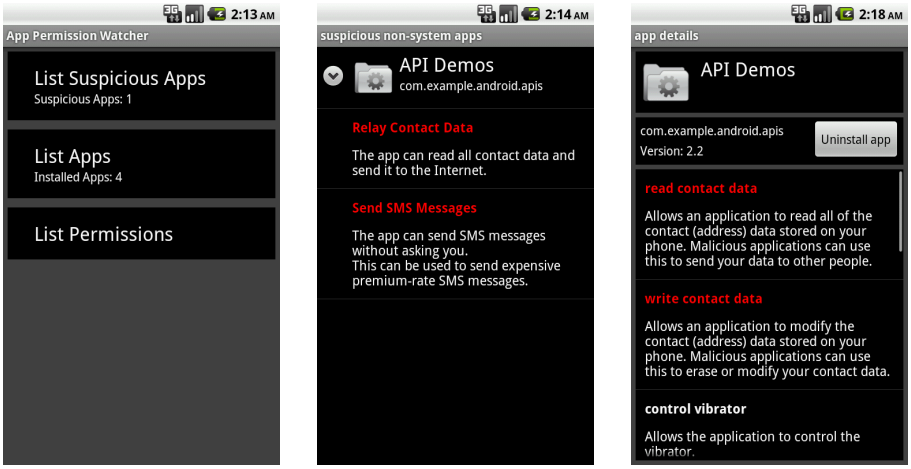
4.3 Application User Interface

The design and implementation of the PermissionWatcher user interface follows the basic principle of *details on demand*, to provide the user with only as much

pieces of information as necessary. Further, we designed the application to integrate seamlessly in the Android system.

After launching PermissionWatcher (either using the home screen widget or the default launcher), the main screen is presented (see Figure 2(a)). Here the user can select three different options: 1) reviewing which applications are detected as suspicious, 2) reviewing all non- system applications, and 3) reviewing a list of all permissions and which applications are using them.

Figure 2(b)) shows the list with the suspicious applications (in this case only one application is contained) while details regarding rules for responsible marking the application as suspicious are expanded. When a user performs a *long touch*, the application details view is started (see Figure 2(c)). Here the user can uninstall the application by pressing the corresponding button. Further, all details about the permissions are provided.



(a) The main view of PermissionWatcher providing three options: suspicious apps, all apps, and all permissions.

(b) The suspicious applications list shows apps and corresponding permissions.

(c) Application details are summarized in application detail view. Users can uninstall an suspicious apps from here.

Fig. 2. Screen shots of the PermissionWatcher graphical user interface.

5 Empirical Usage Evaluation

To gain in depth insights how users make use of the presented PermissionWatcher application, we conducted a field trial in which we collected rich data about usage patterns. To do so, we published the PermissionWatcher application with a built in user data aggregator and advertised the application via different (inter-

national) email lists. These list have $\approx 3,000$ subscribers and a partial focus on Germany.

5.1 Data Collection

Data collected by the aggregator included phone status information, *e.g.*how many and which suspicious applications were installed. Moreover, data about the PermissionWatcher usage were recorded. For instance, how often PermissionWatcher was launched, which view (UI/screen) was used, which applications were marked as trusted and which were uninstalled. In addition, the aggregator recorded how often context menus were opened.

Users were informed about the data logging process before installation. In order to ensure that data could not be used for privacy violations all data was fully anonymized and encrypted before sending them to the collection server.

5.2 Results

Within 6 weeks after release, PermissionWatcher was downloaded and installed by over 1,000 users. Most users were from Germany (72%), followed by Austria (8%), USA (7%), Switzerland (3%), and the UK (2%). Other countries (*e.g.*Brasil, China, Japan, US) had a share of less than 1%. In the Android Market users rated the application 40 times with an average rating of 4.7 (1=poor, 5=best).

After removing data logs which were not readable, data logs of 1.036 distinct users remained containing usage data of 3,669 usage sessions. That is, the application was used in 3.54 sessions (Mdn=2) on average per installation. About 400 users used the PermissionWatcher at least three times within the period of data collection. The duration of the sessions was in 60% of the cases less than 60 s (Mdn=33.5 s).

Users of PermissionWatcher had on average 53.6 applications installed on their Android phones (Mdn=43; SD=45.6). Among these, 10.31 (Mdn=8; SD=9.6) were identified as suspicious by PermissionWatcher which corresponds to 20% of all applications installed on the users' phones. The top five list of suspicious apps installed (number of installations) is shown in following table 2.

Table 2. The top five of most frequently installed applications that were identified as suspicious.

Application Name	# Installations
Whatsapp	476
Google translate	328
Barcode scanner	309
Facebook	292
Skype	239

The 25 most frequently installed apps that were identified as suspicious were each installed on at least 10% of the users. The most frequent reason for tagging an application as suspicious was permission to read and relay the address book information (READ_CONTACTS and INTERNET).

We found that 94 (9.1%) of the users uninstalled at least one application using PermissionWatcher. On average 3.1 applications were uninstalled (Mdn=2). Three users uninstalled more than 10 applications. In total 247 different applications were deleted, whereas only 26 were deleted by multiple users. The top five list is shown in table 3. Note that each application was uninstalled in four cases.

Note: In our initial survey, 44% of all users indicated that they would deinstall applications if they were warned. However, it is well known that people tend to give "socially expected" answers. In addition, our initial survey was even more biased towards security professionals then the field trial; this could also explain a part of the gap.

Table 3. The top five list of most frequently uninstalled applications using PermissionWatcher.

Application Name	# Uninstallation
Compass	4
Ebay	4
Barcode scanner	4
HRS hotel search	4
Skype	4

In more than half of the cases, users were reviewing the list of *suspicious apps* before uninstalling applications (55%). In 29% of the cases the list *all apps* preceded the uninstallation and in 16% the list of *all permissions* was reviewed.

Users had the opportunity to mark suspicious applications as trusted, which removed them from the corresponding list. Only 47 users marked applications as trustworthy (137 in total) which were previously detected through the system. Each of these users marked 5.7 (Mdn=3; SD=6.3) applications as trusted. The top five list applications that were marked as trusted (# cases) using PermissionWatcher is listed in table 4.

Turning towards which features of the application were chosen, the overview list *suspicious apps* was used by 92% of the users. That is, each user opened this list on average 3.2 times (Mdn=2). On average, the usage duration was 60 s (Mdn=19 s; SD=34.4). The overview list *all apps* was used by 45% of the user and was opened 1,077 times. That is, on average each user used this list for 1.9 times (Mdn=1, SD=2.4). In this case, users spend on average 54 s (Mdn=13.1 s; SD=436.9). The list giving an overview of *all permissions* was used by 70% of the users. In total 1,171 times, which is on average 1.6 times per user (Mdn=1; SD=1.5). Users spend on average 83 s using this list (Mdn=32 s;

Table 4. The top five list of applications that were marked as trusted.

Application Name	# Trusted
Whatsapp	11
Skype	10
DB Navigator	9
Google+	8
AVG Antivir	7

SD=274.7). Surprisingly, only 16% of the users viewed the *application details* overview. In total 513 times were application details reviewed which corresponds to 3.1 times on average (Mdn=2) per user. On average, when reviewing the application details, users spend 16.8 s (Mdn=9.3). Figure 3 gives an overview of how many percent of the users applied which data view of PermissionWatcher. This correlates to our finding that users prefer easy to understand presentation rather than the full information. On the other hand, we think that this feature gave credibility to PermissionWatcher.

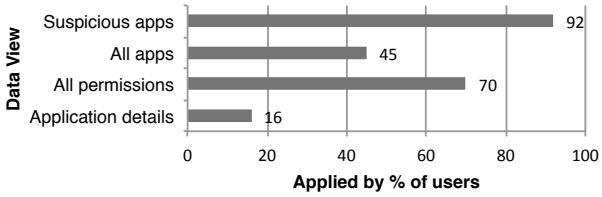


Fig. 3. Usage of different features of PermissionWatcher

In summary, we observed over 2,200 installations within a time period of six weeks. We collected data from over 1,000 users. It appeared that only free applications were considered as suspicious. Many of them were listed in the Android Market top application lists. The main reasons for identifying applications as suspicious are access to the address book or access to GPS sensor data. Interestingly, lists of deleted and trusted applications are overlapping. It is noteworthy that 18 applications that were uninstalled by users were removed later on from the Android Market due to unknown reasons.

6 Conclusions & Future Work

Our user study has indicated clearly that users are actually *willing* to use secure applications as long as the information is presented in an easy understandably way. The most prominent example is the right *full internet access*, which only

63% of all participants in our initial survey could give (*cf.*sect. 3). This points to a serious security problem: if users cannot understand the permissions they grant they are likely to allow Trojan software directly through their front door by clicking at dangerous permission sets. On the other hand, our initial survey also indicated that users *wanted* security on their phones—as long as it was presented in a clear and understandable way.

Therefore, we have developed PermissionWatcher especially for users without a technical and security background. Our aim was to put permission based systems on a stable footing by informing users about dubious permission sets. Results of a field study with 1000+ users indicate that a considerable amount of users carefully examined *suspicious* apps and even uninstalled them. Due to the field study design, it was not possible to contact users and interview them which were reasons for uninstalling applications. Even more interesting, what were reasons for keeping applications that are clearly requiring too many permissions. This needs to be investigated in followup users studies.

There are several ways to extend the concept of creating awareness after applications are installed. First, it would be beneficial if the user would be informed *before* installation about suspicious applications. Preferably, the user should be given alternatives, such as "This torch light app needs 124 permissions, including 15 dangerous ones. Alternatively, we have found a torch light application, that only needs 2 permissions, none of them being dangerous. Which one would you like?"

PermissionWatcher does not use the "wisdom of the crowd" yet. In particular, when choosing alternative applications or identifying dangerous permission sets, users could rely on (indirect) input from others. Judging from our previous experience, this needs to be done in a transparent, and in particular none-interrupting way.

Finally, it would be highly beneficial for the security of the Android Marketplace if alternative suggestions and highlighting problematic permissions were *directly* integrated into the Marketplace itself: instead of *pretending* to offer the user security by showing a mostly unintelligible heap of permissions, it needs to be far more user friendly. Otherwise, large portions of Android users will easily fall prey to attacks by seemingly innocent permissions. On the up-side: The presentation of all permissions needed for one app made it possible for us to identify dangerous permission sets quickly. So the Android market place is certainly on the right track—but would need to take this little extra step to be beneficial to *all* users.

References

1. T. S. Amer and J. B. Maris. Signal words and signal icons in application control and information technology exception messages – hazard matching and habituation effects. Working Paper Series 06-05, Norther Arizona University, 2006.
2. Jonathan Anderson, Joseph Bonneau, and Frank Stajano. Inglorious installers: Security in the application marketplace, 2010.

3. David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *CCS '10*, pages 73–84, New York, NY, USA, 2010. ACM.
4. Michael Becher, Felix C. Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In *SP '11*, pages 96–111, Washington, DC, USA, 2011. IEEE.
5. Serge Egelman, Lorrie Faith Cranor, and Jason I. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *CHI '08*, pages 1065–1074. ACM, 2008.
6. William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proc. of the 20th USENIX Security Symposium*, 2011.
7. William Enck, Machigar Ongtang, and Patrick McDaniel. Mitigating android software misuse before it happens. Technical report, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, 2008.
8. William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *CCS '09*, pages 235–245, New York, NY, USA, 2009. ACM.
9. William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE Security and Privacy*, 7:50–57, January 2009.
10. Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *CCS '11*, pages 627–638, New York, NY, USA, 2011. ACM.
11. Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *WebApps'11*, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.
12. Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. Technical Report UCB/EECS-2012-26, University of California at Berkeley, 2012.
13. Google Inc. Security and permissions. `http://developer.android.com/guide/topics/security/security.html`, 2011. (Last access: 2012/06/20).
14. Giles Hogben and Marnix Dekker. Smartphones: Information security risks, opportunities and recommendations for users. Technical report, ENISA, 2010.
15. Bo Li and Eul Gyu Im. smartphone, promising battlefield for hackers. *Journal of Security Engineering, South Korea, ISSN : 1738-7531*, 8:89–110, February 2011.
16. Microsoft Corporation. Microsoft security intelligence report, volume 11. `http://www.microsoft.com/security/sir/default.aspx`, 2011. (Last access: 2012/06/20).
17. R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *NDSS '11*, pages 17–33, San Diego, CA, 02/2011 2011.
18. Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. Towards formal analysis of the permission-based security model for android. In *ICWMC '09*, pages 87–92, Washington, DC, USA, 2009. IEEE.
19. Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. A formal model to analyze the permission authorization and enforcement in the android framework. In *SOCIALCOM '10*, pages 944–951, Washington, DC, USA, 2010. IEEE.