A Survey on the Evolution of Privacy Enforcement on Smartphones and the Road Ahead

Jan Pennekamp, Martin Henze*, Klaus Wehrle

Communication and Distributed Systems, RWTH Aachen University, Germany

Abstract

With the increasing proliferation of smartphones, enforcing privacy of smartphone users becomes evermore important. Nowadays, one of the major privacy challenges is the tremendous amount of permissions requested by applications, which can significantly invade users' privacy, often without their knowledge. In this paper, we provide a comprehensive review of approaches that can be used to report on applications' permission usage, tune permission access, contain sensitive information, and nudge users towards more privacy-conscious behavior. We discuss key shortcomings of privacy enforcement on smartphones so far and identify suitable actions for the future.

Keywords: Smartphones, Permission Granting, Privacy, Nudging

1. Introduction

Recent years have shown a tremendous increase in the worldwide adoption of smartphones [1]. According to the market analyst International Data Corporation, more than 340 million new smartphones are sold every quarter [1]. In contrast to traditional mobile phones, smartphones can provide users with more features, as they are equipped with various sensors and usually provide a continuous Internet connection [2, 3]. Still, the limited resources of smartphones, especially with respect to computational power, storage space, and energy, call for an outsourcing of computation and storage extensive tasks, most notably to cloud services [3, 4]. As a result, smartphones are frequently used in combination with cloud services nowadays to have data available everywhere and at all times [5].

At the same time, the mobile operating system market is virtually completely shared between Android, iOS, and Windows Phone [1]. While the Android system is mostly open-source and as of 2017 has the largest market share of shipped units (85%), iOS and Windows Phone as proprietary systems achieve a market share of 14.7% respectively 0.1% [1]. All these mobile operating systems have in common that they allow users to install third-party applications, i.e., software that is not necessarily tested in any kind by the smartphone manufacturer or network operator. Usually, this software is obtained through an official store, such as the Play Store in Android or the App Store in iOS [6]. These stores allow a user to simply install an application on her device to extend its functionality. While third-party applications offer enormous benefits to the user, they can also be the cause of severe privacy problems, as the user does not know how they behave. As these applications typically have permission to access the Internet, they can upload all accessible private information of a user to remote servers, often located in the cloud. For example, when first launching the application, WhatsApp transfers the phone's contact list to its cloud servers [7].

*Corresponding author. Tel: +49-241-80-21425, Fax: +49-241-80-22222
Postal address: Informatik 4 (COMSYS), RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany
Email addresses: jan.pennekamp@rwth-aachen.de (Jan Pennekamp), henze@comsys.rwth-aachen.de (Martin Henze), wehrle@comsys.rwth-aachen.de (Klaus Wehrle)

Authors' version of a manuscript that was accepted for publication in *Pervasive and Mobile Computing*. Changes may have been made to this work since it was submitted for publication. Please cite the published version:
http://dx.doi.org/10.1016/j.pmcj.2017.09.005

Notably, protecting privacy is a more complex and delicate challenge on smartphones compared to traditional computers. First, smartphones possess a large variety of different sensors that can be used to monitor and track users in detail [8]. For example, a GPS sensor can accurately report the position of a user. Second, the usage frequency has increased in comparison to traditional computers. Nowadays, a user interacts with her smartphone throughout the day [8]. Hence, protection of privacy and leakage of private data are points that address everybody. Modern computing power allows to process a large amount of information in real-time, i.e., various data sources can be combined to generate a complex user profile [5]. For example, a messenger might track from where and with whom a user is communicating. As more and more tasks, such as shopping, maintaining the calendar, or tracking individual health, are performed with smartphones [5], private data becomes more valuable and worth protecting. On smartphones, the gathered information is more detailed and extensive than ever before and this data reveals an alarming amount of private information on the respective user.

This valuable information is one important reason why applications disrespect the user's privacy [9], as companies are willing to invest more money into personalized advertisement than presenting the same ad on every device, thus reaching their target audience with a higher probability [9]. Due to a large amount of competition, developers of smartphone applications often offer their applications for free and instead present advertisements to users during usage [10]. A recent example is *Amazon Underground*, an application store for Android, which offers all applications for free[1]. In this setting, the developer is paid for the duration her application is being in use. At the same time, Amazon is utilizing the user's usage data, i.e., in other words, her privacy, to improve recommendations for the user [12].

Hence, it is of utmost importance to consider the protection of this private information to safeguard it against third-parties. However, companies aim to utilize this information to increase their revenue. Mobile operating systems try to protect their users by restricting data access and by applying various concepts of information security, such as access control, the principle of least privilege, and sandboxing [13]. However, these measures mostly target malicious applications. Privacy invading applications are not (yet) classified as malicious, and hence, the user's privacy is not significantly protected by today's mobile operating systems [6, 14]. This leads to a situation where the user essentially is surveilled by applications running on her own smartphone [5]. As discontinuing the usage of smartphone applications is neither a feasible nor a realistic solution, the most reasonable approach is to restrict access of applications to sensitive information on the mobile system [8]. Ideally, such functionality should be provided by the mobile operating system itself. Indeed, current mobile operating systems already offer this functionality through a permission management system [6]. However, the user's capability to protect her privacy with such a system is severely limited. We observe that the available options are either not fine-grained enough for reasonable privacy enforcement or do not allow for repeated adjustment, e.g., when a user's perception of privacy evolves [15]. Therefore, researchers and developers propose several implementations and concepts to extend existing embedded permission management systems and to overcome their shortcomings (e.g., [5, 16]). Furthermore, due to the fact that users tend to be lazy, nudging [17, 18] employs different mechanisms to trigger user interaction with the goal of a more privacy-friendly system configuration.

In this paper, we review the evolution of privacy enforcing on smartphones and provide an insight into expected future developments. Through explicitly taking past development in privacy enforcing on smartphones into considerations, we do not only derive a solid reasoning why current mobile operating systems implement privacy enforcement the way they do but also can extrapolate which of the proposed more advanced privacy enforcement systems are most likely to actually be integrated into mobile operating systems. More specifically, our contributions in this paper are as follows:

1. We rigorously analyze the development of privacy enforcing solutions on smartphones, group them based on different levels of privacy enforcement, and extensively compare their strengths and weaknesses. We show that no solution is suitable for properly enforcing privacy and at the same time offering reasonable usability to users.

[1] In April 2017, Amazon announced to discontinue the *Amazon Underground* program at the end of 2019 [11].

2. Additionally, we extract key findings of recent studies related to nudging privacy on smartphones, an approach that has the goal to inform and remind inexperienced users about their privacy.
3. Based on this, we identify shortcomings and challenges of privacy enforcement on smartphones which can be categorized into the solution's usability, users' ignorance, and conceptual problems.

The remainder of this paper is structured as follows. We present the concept of permission management and its implementations in mobile operating systems in Section 2. In Section 3, we introduce more advanced solutions to enforce privacy on smartphones. Subsequently, we present a wide range of approaches to nudge privacy on smartphones in Section 4. We identify shortcomings and open challenges of privacy enforcement on smartphones in Section 5 before we conclude this paper in Section 6.

2. Permission Management on Smartphones

Most functionality, i.e., resources, of a smartphone is protected by APIs for security purposes. Hence, access to this functionality is only granted with the corresponding *permission*. For example, the user's location or the user's contact list are protected and can only be read by applications that have the appropriate permissions. This mechanism protects against misuse of any kind. During development of an application, the developer has to declare all required permissions, which allows the user to verify whether an application asks for unnecessary resources. This mechanism of permission management is called permission-based access control [6]. Without such system, a malicious developer or application could access and hence exploit all resources of a smartphone.

Other important security principles in mobile operating systems include application isolation, i.e., different applications cannot directly interact with each other, and application signing, i.e., the authenticity of an application's source code and the identity of the developer can be verified by the mobile operating system [6]. Recently, Android added SELinx, which aims at protecting against malicious applications, complementary to the existing sandboxing mechanism. In the following, however, we will focus on the permission management system, because this is the part of the operating system the ordinary user can influence to protect her privacy by granting and restricting applications' permissions [19]. Current mobile operating systems do not offer a dedicated interface for privacy control so far, instead, they are designed with respect to enforcing security. As mentioned in Section 1, smartphones run different mobile operating systems and their implementation of the permission system differs. However, the underlying principles are similar. In the following, we first present different *types of permissions* in Section 2.1 and then introduce measures for *grouping of permissions* that should simplify permission management for the user in Section 2.2. Finally, in Section 2.3, we illustrate the user's interaction with the permission system through *granting permissions*.

2.1. Types of Permissions

All kinds of possible actions a smartphone can perform are supported through APIs, such that application developers are able to utilize this kind of functionality. Permissions protect "risky" actions, i.e., an application can only use this API if the matching permission has been granted [6]. If the permission has not been granted, the application cannot access this functionality. In this case, the developer ideally anticipated this situation and the application behaves gracefully nonetheless, i.e., it does not crash. The different types of permissions include a variety of actions [20]: Changing customization settings on the device, making calls, accessing SMS messages, using sensors, accessing the Internet, or using any radio interface. The available permissions depend on the mobile operating system and the device's hardware capabilities. For example, a smartphone without a camera cannot provide access to it. Furthermore, even extremely critical functions, such as changing the device's power status or performing a factory reset of the device, are supported through API calls [20].

In general, mobile operating systems demand from the application developer to statically list the required permissions in their application, hence, virtually everybody has access to this information. Currently, Android offers 135 unique *fine-grained* permissions [21]. Notably, even system applications make use of the permission system in Android. For this reason, some dangerous permissions, such as wiping the device, can

Permission Groups
CALENDAR
CAMERA
CONTACTS
LOCATION
MICROPHONE
NORMAL
PHONE
SENSORS
SMS
STORAGE

(a) Permission groups defined in Android to combine fine-grained permissions.

Permission Name	Permission Details
READ_PHONE_STATE	Read-only access to phone state
CALL_PHONE	Initiate phone call to any number
READ_CALL_LOG	Read user's call log
WRITE_CALL_LOG	Write-only access to user's contacts
ADD_VOICEMAIL	Add voice mails to the system
USE_SIP	Use SIP services (IP telephony)
PROCESS_OUTGOING_CALLS	Access to the number dialed during a call

(b) List of permissions contained in the permission group *PHONE*.

Table 1: Android groups permissions into ten permission groups which each consist of a larger number of individual fine-grained permissions [21].

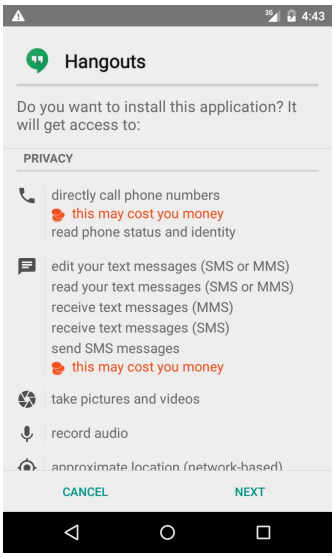
only be granted to system applications. The granularity of the permission system has an impact on the trade-off between simplicity for the application developer, on the one hand, and security, on the other hand [19]. A coarse-grained system grants access to more functionality through a single permission, while a fine-grained system requires the developer to meticulously list all required permissions. Representing permissions with very fine granularity might still lead to serious security threats, as the combination of multiple permissions can pose new threats [6]. For example, while only allowing access to the microphone or the Internet for an application individually could be considered harmless, allowing access to both is far more dangerous. In this case, an application, for instance, would be able to monitor phone calls and transfer them to a remote server in real-time. Recently, Fratantonio et al. [22] identified severe security issues originating from the combination of permissions and related design shortcomings in Android. Still, a permission system is a reasonable approach to limit the capabilities of applications on a system that offers support for third-party applications.

2.2. Grouping of Permissions

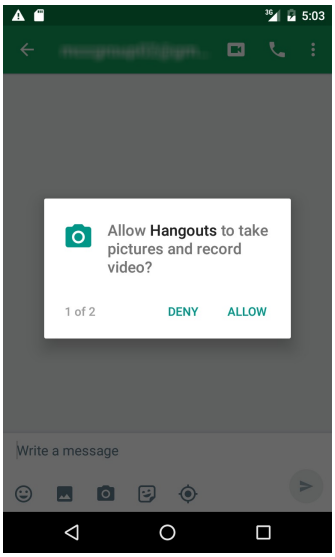
Modern smartphones offer a lot of functionality and, therefore, a high number of permissions is required to cover all of it. However, ordinary users might not be able to understand and differentiate between a large number of permissions, and for that reason, they might feel overwhelmed [19]. The main idea to support those users is to group similar permissions together. Hence, a fine-grained representation of permissions is grouped into a more coarse-grained categorization based on criteria, such as requested access or risk level for the user [21]. For example, all functionality related to the camera can be grouped into one group, while all permissions related to the user's location might be grouped in another.

Table 1a illustrates Android's ten permission groups [21]. For example, the categories differentiate permissions related to the calendar, the camera, or the location. The permission group *NORMAL*, which contains permissions such as "VIBRATION" or "ACCESS_NETWORK_STATE", is unique, because those permissions are automatically granted to installed applications. The coarse granularity of grouped permissions is intended to support the user's decision making [20]: On the one hand, a user might understand the meaning of the permission group *SMS*, while the permission "RECEIVE_WAP_PUSH" is too specific to understand. On the other hand, a user might not expect specific permissions in a group, as, for example, the "RECEIVE_MMS" permission is part of the permission group *SMS*. For clarity and as an example, Table 1b lists all permissions contained in the permission group *PHONE* along with a short description of each permission. Permission groups that include permissions which might cost the user money, for example, *PHONE* or *SMS*, are annotated in an eye-catching way to make the user aware that granting access to this functionality to an application might result in additional costs for the user (see Figure 1a) [13]. So far, Android does not support annotations for other possibly negative effects, such as a high battery consumption or privacy intrusive permissions.

The number of permission groups in other mobile operating system is slightly higher when compared to Android. Apple's iOS outlines 19 *coarse-grained*, so called, capabilities [23], while Windows Phone offers



(a) The permission install prompt in Android < 6 lists all permissions an application requests at install time.



(b) The permission runtime prompt in Android ≥ 6 asks for a specific permission when it is first requested.

Figure 1: Granting permission can be either granted at install time for all permissions an application might ever need or during runtime when a specific permission is first required.

35 *coarse-grained* capabilities [24]. In contrast to Android, these systems do not provide a fine-grained representation of the permissions in a permission group.

2.3. Permission Granting

Granting and denying permissions is an essential part of a permission system, because otherwise all access requests would simply be granted and the system would be rendered useless. Therefore, in current systems, the user has the ability to grant or deny permissions. Ideally, she should be able to adjust her decision afterward. Two concepts are implemented in common mobile operating systems to prompt the user for granting permissions [6, 19], *install time granting* and *runtime granting*, which we both illustrate in Figure 1.

First, *install time granting* presents a dialog to the user to confirm the requested permissions when installing an application. We provide an example in Figure 1a. The system presents an overview of all permissions asked by the application. Usually, the user can only accept all of them or decide to not install the application at all. Install time granting is implemented by Android and Windows Phone. Second, *runtime granting* prompts the user when an application tries to use a permission during runtime. In this case, each permission is requested on its own whenever it is needed. Hence, the user might be prompted repeatedly by one application. However, the user can grant specific permissions, while denying others. Figure 1b includes one example prompt. The operating system usually remembers the decision, i.e., the user is only prompted once per application for each individual permission. Runtime granting is supported by iOS and Android starting from Version 6.

To reduce complexity in both approaches of permission prompting in Android, the system is simplified as far as possible. Permissions that are included in the permission group *NORMAL* (check Section 2.2) are automatically granted to applications [21], because they are considered to be non-dangerous. Hence, the user is not prompted for them and might not even be aware that they have been granted. In case the user installs an application update that requires additional permissions, she is only prompted, if the application requests a permission from a previously not requested group [6]. For example, an update to a calendar application that wants to obtain access to the user's contact list to implement an invite feature would result

in an additional install time prompt to the user for the newly requested permission group if a permission of the group *CONTACTS* has not been granted previously. Permissions in a group in which another permission had been granted previously are automatically granted. For example, an application that has the permission to call a support number (Permission "CALL_PHONE") and, therefore, with access to the permission group *PHONE* (check Table 1b), can also read the user's call log in an updated version without an additional install time prompt. This issue further exacerbates when considering *third-party* permissions [25] or updating the mobile operating system itself [26]. Likewise, in Windows Phone, only new capabilities are prompted during an update [6]. Consequentially, in both systems, the user has an incomplete view of all permission requests during the update process. In iOS, no such policy applies, because additional permissions are requested during runtime when they are first needed.

3. Privacy Enforcement on Smartphones

Privacy is an important challenge in the context of smartphones [5], as a large amount of sensitive data can leak from the device. Hence, users interested in privacy enforcing want to use the best method available to enforce their privacy. For this reason, we deliberately refrain from presenting static analysis approaches in this paper. These approaches are well known from the analysis of malware [27] as well as privacy-related behavior of applications [28, 29] and existing tools support manufacturers to improve their security policies [30]. Additionally, static approaches have even been used to realize a permission removal system to protect the users' privacy [31]. However, as static analysis performs a theoretical evaluation, it is prone to non-existing data flows which might lead to the identification of only theoretical privacy leaks. When enforcing privacy on smartphones, we are not interested in theoretical problems but rather real-world privacy risks. Furthermore, static analysis tools typically do not target end users and, hence, inexperienced users might not be able to utilize them to properly constrain their privacy. Hence, we focus on dynamic and passive approaches in this paper, as these work in practical scenarios with real user interaction. While these approaches allow users to enforce their privacy on smartphones, they typically come at the cost of an increased computation overhead on the devices, hence, decreasing devices' battery life.

In the following, we present a comprehensive overview of approaches that make use of the permission granting system to enforce privacy on smartphones. As current mobile operating systems lack important aspects of protecting users' privacy, a wide range of such approaches is available. For example, current mobile operating systems do not offer support for customizing permissions or their level of privacy enforcement is not sufficient. We assume that once an application with permission to access the network obtains sensitive data, this data *can* be transferred from the device to any remote location. This becomes especially problematic if data is transferred to third-party cloud services, as the user suffers from non-transparency as she is unaware of the location of the data storage, the implemented security mechanisms, and the jurisdiction that applies to her data [32]. To inform the user about the privacy risks she is facing if cloud services gain access to her data, PrivySeal [33] visually presents resulting potential privacy leaks and MailAnalyzer [34] uncovers the exposure to cloud services while sending and receiving email. In this paper, we deliberately do not study privacy approaches that rely on a server component (e.g., AntMonitor [35] or Windows Phone Tracking [36]), as they induce further privacy and deployment issues for the ordinary user. We categorize the presented approaches with respect to the *level of user manipulation*, *amount of privacy enforcement*, *level of system modification*, *usability*, and whether the approach is *open-source*.

For the *level of user manipulation*, we identify three categories which we use in the following: *Reporting* (Report), *Fine-grained Tuning* (Tune), and *Fencing Information* (Fence). To allow for a better comprehension, we illustrate the connection between the level of user manipulation and the amount of privacy enforcement in Figure 2. Approaches that only have a *reporting* nature simply inform the user about granted or denied permissions (cf. Section 3.1). Hence, users can only interact with these approaches in a very limited fashion. Nevertheless, a reporting functionality is useful to provide users with a well-founded basis for their decision about application usage. Second, approaches that also allow the user to selectively toggle permissions on the smartphone are labeled as *fine-grained tuning* (cf. Section 3.2). These approaches follow the reasonable assumption that an application's access to specific functionality, i.e., sensitive information, is either permitted or restricted and no graduation between the two exists. Finally, more advanced

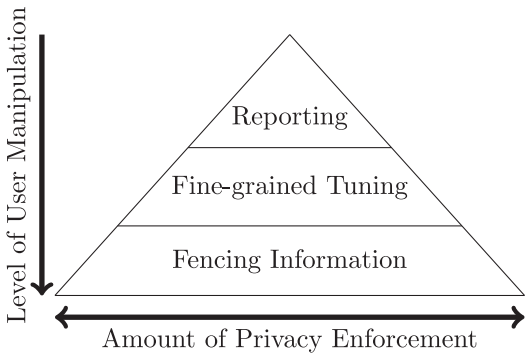


Figure 2: A higher level of user manipulations leads to increased privacy enforcement.

implementations for *fencing information* enable the user to set up more mature privacy boundaries (cf. Section 3.3). For example, applications only get access to forged data or the operating system restricts network access to prevent leakage of sensitive information from the smartphone to cloud services. Consequentially, the more possibilities the user has, the more powerful her privacy enforcing can be.

We provide an overview of all approaches covered in this paper and their specific implementations in Table 2. When comparing these approaches, the *level of modification* to the operating system is an important aspect, as it determines how widely the approach can be used. We differentiate between no modification (-), root user access (root), and the need for a modified system image (image). *Root user access* is often required to overcome the boundary of application isolation which is implemented for security purposes in current mobile operating systems but might be an obstacle for privacy controlling applications. Without root user access, an approach cannot influence other applications or access their data, i.e., not control the user's privacy. Unfortunately, granting root user access can lead to security risks [13]. Furthermore, certain solutions need to manipulate information on a lower level, i.e., they rely on functionality the operating system does not export through dedicated APIs. In these cases, it might be necessary to even modify core components of the operating system [5, 6]. To this end, the user has to modify the device's firmware, i.e., flash a *modified system image*. A modified system image is a large obstacle for the user. First, each version of an operating system has to be separately adjusted to deal with changes in the implementation. Even experienced users usually do not have knowledge about these details and the required steps to prepare their device. Second, a modified system image requires an unlocked bootloader, because locked bootloaders only allow the installation of unmodified images for security reasons [6]. To overcome the installation obstacles introduced by these steps for ordinary users, Chainfaire presented an experimental approach for systemless root, i.e., no modifications to the system partition are necessary [37]. This has various additional advantages for the user, e.g., the ability to still receive over-the-air updates. Based on this approach, even more, sophisticated functionality could be provided in a systemless way, which usually requires a modified system image, e.g., Xposed [38] (cf. Section 3.3). Overall, if systemless root would become more widespread, users could benefit from more options to enforce their privacy without trading them for usability.

In our comparison, we rate the *usability* of an implementation for the average user from one to five stars. While we aim to assess the usability of individual approaches as objective as possible, our assessment likely is biased by subjective perceptions. The question whether the presented approach is *open-source* is important as well, because if the behavior of the approach cannot be verified by security experts, the approach for privacy control itself could easily leak sensitive information [6]. Users have to trust the approach they utilize to configure their privacy settings. As most major mobile operating systems are not open-source, most research, and hence also our comparison, focus on Android. Nevertheless, the theoretical concepts presented in this paper could also be applied to other mobile operating systems. Indeed, first approaches have already been presented for other platforms. As an example, PiOS [39] can be used to detect privacy leaks in iOS applications.

Level of User Manipulation		Approach	Implementation	System Modification	Open-source	Usability
	Reporting	Permission Install Prompt	Android [20] / Windows Phone [6]	-	✓/ ✗	⋆ ⋆ ⋆ ⋆
		Permission Visualization	Android [20]	-	✓	⋆
			F-Secure App Permissions [40]	-	✗	⋆ ⋆ ⋆
		Tracking Privacy Information	TaintDroid [5] / NDroid [41] / TaintART [42]	image	✓	⋆ ⋆
			Haystack [43] / Securacy [3]	-	(✓)	⋆ ⋆ ⋆ ⋆
			Uraine [44]	-	✗	⋆ ⋆ ⋆ ⋆
	Tuning	Permission Runtime Prompt	Android [20] / iOS [6]	-	✓/ ✗	⋆ ⋆ ⋆ ⋆
			RecDroid [45]	image	✗	⋆ ⋆ ⋆
		Toggling Permissions	Android [20] / AppOps [17]	varying	✓	⋆ ⋆
			LBE Security Master [46]	root	✗	⋆ ⋆ ⋆
			AppGuard [47] / ARTist [48] / Capper [49]	-	✗	⋆ ⋆ ⋆
		Toggle Install Prompt	Poly [16] (part of Apex) / SDroid [50]	image / -	✓/ ✗	⋆ ⋆ ⋆ ⋆ ⋆
		Trusted User-interface	LayerCake [51] / AUDACIOUS [52]	image / -	✓/ ✗	⋆ ⋆ ⋆ ⋆
	Fencing	Faking Information	AppFence [53] (requires TaintDroid)	image	✓	⋆ ⋆
			XPrivacy [54] (requires Xposed [55])	image	✓	⋆ ⋆ ⋆
			MockDroid [56] / TISSA [57]	image	✓	⋆
			ProtectMyPrivacy [58, 59]	image	✗	⋆ ⋆ ⋆
		Situation Aware Restricting	ConXsense [60] (uses FlaskDroid [61])	image	✓	⋆ ⋆ ⋆
			TurtleGuard [62]	unknown	✗	⋆ ⋆ ⋆ ⋆
		Firewall & Host Blocking	AFWall+ [63]	root	✓	⋆
			AdAway [64]	root	✓	⋆ ⋆

Table 2: The different approaches compared in this paper differ in the level of user manipulation they provide, their usability, their availability as open source, and whether they require modifications to the mobile operating system.

3.1. Reporting

Approaches that simply report information on privacy (enforcement) on smartphones can be used as a first building block by users to make an informed decision about their privacy settings. However, applications merely reporting such information do not provide an interface for the user to withdraw permissions, i.e., the user cannot directly contain a potential privacy leak. Instead, the user has to rely on system tools or additional third-party applications for this task. However, given their representative character, reporting applications do not always require root permissions or even deeper changes to the operating system. Unfortunately, not being able to adjust permission settings significantly restricts the user's abilities. Thus, the user might not exercise her "right" to privacy control. Still, she is able to change her usage behavior, e.g., using an application which only leaks location information in a single location or denying the application access to location information at all, thus, preventing the creation of a detailed location profile.

To further illustrate the concept of reporting on permission settings, we exemplarily discuss how a user would interact with a reporting approach with a standard Android application, such as the *Hangouts* application. Here, the user would be informed that Hangouts requests permissions for access to her location and SMS messages. While the user might understand the need for location access to share her location with contacts, she might not understand why the application requires access to her text messages. Depending on the individual approach, she would, e.g., be presented with contextual information of text message access. While providing (extensive) reports on permissions requested by applications, reporting approaches do not provide the possibility for user manipulation, e.g., to restrict requested permissions. In the following, we present three specific approaches that feature reporting functionality.

Permission Install Prompt. As introduced in Section 2.3, one approach for reporting permissions to the user is the permission install prompt. Thereby, the operating system presents the user with a prompt informing the user about permissions demanded by a third-party application prior to installing it [6]. We include an example prompt from Android 5.1.1 in Figure 1a. Unfortunately, Android simply includes a

general permission description, while a real support for decision making is left out. Furthermore, Android groups detailed permissions by default to limit the amount of information presented to the user [21]. Apart from Android, Windows Phone also prompts the user on install time for applications' permissions [6].

An advantage of the permission install prompt is that all permissions are presented at once, preventing bad surprises later on. A centralized approach to present all permissions helps experienced users to take a well-founded decision. However, with a high number of permissions, this approach also lacks clarity and might overwhelm (inexperienced) users. Additionally, users might not understand why a specific permission is required, as no context of its use is given in the prompt. Even inexperienced users are able to recognize and relate permission install prompts as their representation occurs in an unmodified form, i.e., the interface is identical independent of the application that is being installed. Hence, this concept suffers from the risk that users blindly accept all prompts, as all prompts look alike. Another disadvantage is that users are only prompted once, hence, they might not repeatedly deal with their privacy settings as they possibly should. In this approach, users are not able to selectively grant permissions which constitutes a disadvantage. Rather, they are forced to grant all requested permissions or completely abort the installation of an application.

Permission Visualization. In case a user wants to review granted permissions after install time, she has to rely on another approach as permission install prompts do not offer this functionality. To this end, dedicated applications exist that visualize granted permissions on a per application basis. Android features a basic permission visualization that can be separately accessed for each application [20]. Advanced implementations are available in the Google Play Store that provide grouping and filtering mechanisms for advanced security and privacy analysis. For instance, a user can list all applications that have access to location information. One example is the *F-Secure App Permissions* application[2] [40]. The fact that this application is not open-source is not too alarming as it is not able to modify critical settings due to missing root permissions. Overall, this approach only has informative character, as the user is not able to directly toggle permissions. Nevertheless, it supports users' decision making also after the initial installation of an application. However, without relying on additional software, her options to exercise privacy control are limited to removing the application in question or adjusting her usage behavior.

Frameworks for Tracking Private Information. Both previous approaches use existing functionality of the operating to provide information about granted permissions. As current mobile operating systems do not include an advanced framework for tracking privacy leakage within the operating system, third-party implementations address this shortcoming. As stressed before, extensive modifications to the operating system severely limit the applicability of these tracking frameworks for ordinary users. One example for such a tracking framework is *TaintDroid* [5] which accurately tracks the flow of privacy sensitive information on the device. To this end, each application's behavior is monitored and presented to the user, thereby visualizing the information flow for each application throughout the system. As this requires a modified Android system image, the supported devices are limited to four Google devices and only three Android versions, no later than 4.3 [66]. An improved approach, *NDroid* [41], also enables the analysis of native code, i.e., parts of an Android application written in a language different than Java (e.g., C or C++), and, therefore, achieves more accurate results when compared to *TaintDroid*. For recent Android versions, *TaintART* [42] enables flow tracking in the new Android RunTime (ART) environment, which was introduced in Android 5 and replaces the previously used virtual-machine-based Dalvik approach. Even though these basic implementations do not offer the possibility to restrict access, more advanced projects based on *TaintDroid*, such as *AppFence* [53] (cf. Section 3.3), provide this functionality. A similar approach, *AppIntent* [67], attempts to differentiate between user-intended and unintended sharing of private data between applications. In this approach, unintended sharing of private data is considered as privacy leakage.

Related approaches that do not require a system modification are *Haystack* [43], which recently was renamed to *Lumen Privacy Monitor* [68], and *Securacy* [3]. They utilize the device's VPN interface to monitor the networking traffic for all information that leaves the device and provide users with detailed statistics on privacy leaks of individual applications on their smartphones. In addition, *Securacy* also tries to identify the physical location data is being sent to. *Uraine* [44] takes a completely different approach to

[2]In December 2015, F-Secure announced to discontinue the *F-Secure App Permissions* app due to the permission management improvements resulting from the newly introduced runtime permissions in Android 6 [65].

implement a tracking framework without any system modification. All installed applications are modified prior to their installation on the device such that *Uraine* can log occurring privacy leaks. When compared to *TaintDroid*, this approach introduces a slightly increased computation overhead [44].

3.2. Fine-grained Permission Tuning

In contrast to the above approaches, fine-grained permission tuning allows users to toggle applications' permissions, i.e., to interact with the permission system and not only review permissions. Notably, as not all fine-grained permission tuning approaches automatically present prompts to the user, the user typically has to actively enforce her privacy control by initiating one of the approaches. As fine-grained permission tuning influences operating system settings (granted and restricted permissions), it either requires root user access or a modified system image. Notably, restricting permissions on a system that does not feature permission toggling is problematic, as developers do not always anticipate that requested permissions might not be available. Hence, applications are likely to crash as a result of missing permissions [13]. In the following, we present four approaches that utilize permission tuning to increase the level of privacy enforcement on smartphones.

Again, we illustrate this approach by referring to Android's *Hangouts* application. Using fine-grained permission tuning, the user could decide which permissions to grant and which permissions to withdraw. For example, she could grant the application location access while permitting access to her contacts. In practice, this would not have any negative influence to the core functionality (messaging) and the application would behave gracefully. However, if the user decides to withdraw the permission for access to her identity stored on the smartphone, i.e., her stored accounts, the application would crash as this access is required to provide the messaging functionality. Hence, with these approaches, the level of user manipulation and the available amount of privacy enforcement is improved, while, certain decisions might render the application unusable.

Permission Runtime Prompt. The most basic approach for fine-grained permission tuning is the permission runtime prompt (cf. Section 2.3). Whenever an application requires a permission, the operating system will prompt the user for granting this permission. The user can then decide whether the requested permission is necessary and acceptable at this point in time. This approach is implemented in iOS [6] and Android 6 [20]. The user interface between these two implementations does not differ significantly and we include an exemplary Android runtime prompt in Figure 1b. *RecDroid* [45] extends these basic permission runtime prompts with a recommendation given by experienced users. To this end, *RecDroid* queries a crowd-sourced database on each permission request. This approach requires a modified system image and only supports Android 4.3 at the moment.

In contrast to install time prompts, the user can make more informed decisions as she is aware of the context of permission requests. Thus, she might more easily understand the consequences of her choice. Furthermore, the user is explicitly educated about permission requests demanded by used applications. Notably, users are always presented with unmodified permission runtime prompts to allow even inexperienced users to recognize and relate them independent of the application in use. Therefore, permission runtime prompts suffer from the risk that users blindly accept all prompts, because they all look alike, might occur repeatedly for different permissions, and interfere with the user's current action [69]. With respect to prompting strategies, we differentiate between two approaches [20]: Either the user is prompted repeatedly for each permission or only once for each permission, i.e., her decision is remembered. If the prompt is shown repeatedly, the user might either be annoyed or glad that she was reminded to enforce her privacy control. If the prompt is only shown once, she might forget about granted permissions. Unfortunately, in some situations, permission runtime prompts do not work well, e.g., when users want to quickly take a picture or instantly record audio.

Toggling Permissions at Runtime. Although the user can use runtime prompts to configure permissions, she might be interested to actively adjust permissions of applications at any time. This, most notably, includes the revocation for previously granted (permanent) permissions. To this end, both Android 6 and iOS provide a configuration view that presents the demanded and granted permissions for each application [20, 23]. In Android KitKat (4.3-4.4.2), support for such a configuration view, called *AppOps*, was included in the system [20], but it was not accessible to users, i.e., the settings menu was not reachable through the user interface. To still allow access to AppOps in this setting, various applications implemented a shortcut

to make AppOps accessible [70]. The layout of AppOps is comparable to the install time prompt. However, in contrast to the install time prompt, AppOps enables users to modify permission settings at any time. To this end, AppOps allows to revoke granted permissions and grant new permissions. AppOps suffers from the same disadvantage as install time prompts, as the user is not supported through contextual information in deciding on the necessity of certain permissions.

Although the above approaches are limited to specific versions of mobile operating systems, third-party applications offer similar functionality for other versions of mobile operating systems. A well-known application that realizes permission toggling is *LBE Security Master* [46]. It even offers an automatic mode which sets the permissions to settings recommended by the developers of LBE. As *LBE Security Master* is officially available only in Chinese, its popularity led to a large number of custom translations [71]. Because *LBE Security Master* is not open-source, the user has to trust the developers to not breach her privacy. This is especially problematic, as this kind of applications usually requires root permissions to interact with system settings that affect other applications. *AppGuard* [47] circumvents this constraint by instead modifying the installed applications on Android versions up to 4.4 (cf. *Uraine*). To this end, every third-party application is patched to include an interface for the logging and configuration component of *AppGuard*. *ARTist* [48] is the successor of *AppGuard*, which is also compatible with Android versions starting at 5.0. However, so far the implementation of *ARTist* is not publicly available. *Capper* [49] extends this approach by enforcing permission access with context-awareness. The disadvantage of this approach is that all applications have to be re-installed and the embedded modifications are not persistent over updates, i.e., applications have to be patched again after each update.

Toggle Install Prompt. Trying to tackle disadvantages of the basic install prompt, the toggle install prompt (similar to toggling permissions at runtime) allows to toggling specific permissions rather than just granting or denying all of them at install time. This way the user's ability to enforce privacy control is significantly improved, as she can decide for each requested permission whether she wants to grant it or not. Currently, no mobile operating system provides such behavior [6]. Hence, realizing this approach currently requires a modified operating system image. For example, *Poly* [16] realizes a toggle install prompt for Android 2.x. It has been proposed as part of the Android permission extension (*Apex* [16]) framework that supports runtime constraints for permissions. As one example, *Apex* allows to limit the number of text messages an application can send per day. This functionality of *Apex* belongs to the concept of *Situation Aware Restricting*, which we will detail in Section 3.3. In contrast to *TaintDroid* (cf. Section 3.1), *Apex* focuses on actually restricting the usage of resources and not just reporting the flow of sensitive information. Unfortunately, no implementation of *Apex* is publicly available. *SDroid* [50], a more recent approach, combines user toggling with a local database to offer permission recommendations to the user upon application installation. A related approach also incorporates toggling permissions specifically for advertising purposes to improve users' privacy [72]. For example, a user can grant an application access to her location to provide the intended functionality, while she can restrict location access through the same application for advertisement purposes. Similarly, a different approach evaluated the users' willingness to pay for less requested permissions upon install time [73]. However, an answer on how to agree on an automated consent is yet to be found. Such approaches would require a modified operating system image and the patching of applications. Again, no implementation or prototype exists to further evaluate these approaches.

Trusted User-interface. Building upon the concept of the basic runtime prompt, Roesner et al. [2] propose a non-intrusive way to prompt the user for permission. Instead of explicitly asking for permission, they suggest integrating permission granting into the user interface of the individual application. To this end, trusted user-interface (UI) elements are provided by the operating system to implement a large part of the permission system unnoticeable for the user [2]. For example, clicking on a standardized "microphone" button would automatically grant permission for using the microphone to the application that embedded the button. Therefore, this approach is not as intrusive as a usual runtime permission prompt. The advantage of a trusted UI is that the application only temporarily gains access to private data if and only if the user explicitly triggers an action. As the permission granting is implicitly achieved through trusted UI elements, this concept cannot cover the granting of all possible permissions. For example, the processing of incoming text messages is not triggered by users. Hence, a trusted UI cannot be used to grant an application access

to this processing. *LayerCake* [51] is an example for a trusted UI implementation on Android that requires deep adjustments to the operating system, because core functionality needs to be modified to include trusted UI elements. Because of the resulting effort for adapting *LayerCake* to individual hardware and operating system versions, it is only available for two smartphone models, namely Nexus S and Galaxy Nexus, running Android 4.2. To overcome these issues, *AUDACIOUS* [52] realizes trusted UI elements for the Android platform without the need to modify the operating system. This approach is based on a secure library which is combined with static and dynamic analyses. While *AUDACIOUS* does not require support by the operating system, it relies on application developers to include the secure library into their applications and replace standard UI elements with the trusted UI elements provided by *AUDACIOUS*.

3.3. Fencing Information

The approach of not granting (or revoking) permissions requested by an application might not always be feasible. If the developer of an application does not anticipate that permissions might not be granted, the application might crash when requested permissions are not available, i.e., fine-grained permission tuning results in a bad user experience. In contrast to strict permission adjustments, i.e., denying access to requested permissions, the approach of *fencing information* aims at not interfering with the intended functionality of the application. In particular, access to private information is granted, however, the application may not leak it to a remote server. Most notably, when fencing information on the device, application crashes are avoided. Furthermore, this approach does not require any changes to individual applications. For example, an alarm clock application should, from a privacy perspective, work without having access to the location (which the application might use to display local advertisements). There are different ways to fence location information and still use this application: Either the application can be provided with forged or omitted location data by the operating system or flow of this sensitive information outside of the device can be prevented. The approaches we analyze in the following have in common that they either require modified system images or root user access as they rely on the modification of application data.

Faking and Restricting Information. Instead of not granting permissions to applications, users can use solutions that grant applications access to forged or omitted information only. For example, the operating systems could return a manipulated location to the application requesting the current location [53]. Another example is the operating system only providing access to an empty contact list instead of providing real user data [54]. Various API calls, supported in current mobile operating systems, can be modified in such a way to increase users' privacy. *MockDroid* [56], a simple research prototype, was the first approach to present such functionality.

Currently, two Android implementations are available to forge and restrict information. The first example is *AppFence* [53], which is based on the *TaintDroid* framework and uses two approaches to reduce privacy leakage in Android applications. First, the user can instruct the application to replace sensitive information with forged data. Second, the user can restrict which data is allowed to be transmitted over the network, which equals a firewall implementation. As *AppFence* is based on *TaintDroid*, it suffers from the same limitations, such as limited availability, while providing good data tracking abilities. The second implementation, *XPrivacy* [54], provides similar functionality. However, even though *XPrivacy* offers a variety of detailed settings, it is meant to be simple, because information access is restricted by default and the user is prompted for interaction during runtime. In contrast to *AppFence*, *XPrivacy* is based on the *Xposed* framework [55], which modifies the Android system to offer an advanced interface for developers to change system behavior. It is available for a large group of devices [55]. Unfortunately, *Xposed* has to be adjusted for each new Android version (as of July 2017, no version for Android 7, whose first official rollout was in August 2016, has been released [74]). *TISSA* [57] is an Android implementation which is embedded into the operating system in a similar way as *TaintDroid*, while providing comparable functionality as *XPrivacy*. The user can monitor privacy leaks and decide for each application whether to grant access only to anonymized, forged, or omitted data. Unfortunately, no implementation of *TISSA* is publicly available. *ProtectMyPrivacy* [58] provides faking and restricting information for devices running a jailbroken iOS. In particular, the implementation allows the user to replace sensitive private information by anonymized data, e.g., granting access to a contact list consisting of forged users instead of sharing a valid list of contacts. For the sake of usability, it features a recommendation system which provides effective and functional settings

especially for inexperienced users. In addition to an iOS version, an Android implementation of *ProtectMyPrivacy* [59] has been presented, which requires the *Xposed* framework [55] to be installed. However, its overall functionality is not as extensive as *XPrivacy*'s, making it more suitable for inexperienced users.

Situation-aware Restricting. Similar to the previous approach, where access to information is always fenced, situation-aware restricting takes the current situation into account when deciding whether to restrict information access or to grant access to user data. For example, a user might be fine with sharing her location in public locations, while her home or office location should not be shared. This context profiling can be implemented based on either machine learning or explicit user decisions [60]. Unfortunately, sensing for situation awareness accurately in an energy-efficient way is very difficult to achieve [75]. The challenges faced here are comparable to approaches trying to improve device unlocking, such as trusted devices, face unlocking, or location awareness.

ConXsense [60] is one of the frameworks that realize situation-aware restricting. It is able to automatically classify situations regarding their security and privacy properties. The functionality is integrated into Android through the generic security framework *FlaskDroid* [61], which allows adjustments to Android's security architecture in an efficient policy language, but is only available for Android 4.0. *TurtleGuard* [62] is a permission manager that supports situation-awareness for a subset of permissions. In particular, the user can choose between always granting permissions, granting when in use, and always restricting permissions. Furthermore, *TurtleGuard* supports granting access to privacy-sensitive information at different granularities. The authors plan to evaluate their approach as part of a field study. So far, *TurtleGuard* is not publicly available.

Firewall and Host Blocking. Two approaches that do not rely on the permission system of the mobile operating system to prevent privacy leakage are firewalls and host blocking. Their underlying idea is to contain a closed environment and builds upon the approaches of *Haystack* and *Securacy* (cf. Section 3.1). While a firewall restricts network access for applications, host blocking globally prevents access to specific remote servers (e.g., to prevent tracking by advertisement networks). Hence, privacy protection can only be achieved if network access is restricted in all situations.

Various open- and closed-source implementations of firewall and host blocking exist for the Android platform. *AFWall+* [63] implements the firewall approach, while *AdAway* [64] is an example of a host blocking application. Both approaches require root permissions, because they need to be able to manipulate system behavior. As the Android operating system already features a hosts file, and granting and restricting of the Internet permission to individual applications, no system image has to be modified. Applications with similar functionality are available for jailbroken iOS systems (e.g., [76]). To the best of our knowledge, no application implementing the firewall approach for Windows Phone has been presented. In any case, users could manually edit the hosts file of the respective operating system.

Firewalls are well-known from computers and suffer from the same downsides on the portable device, i.e., overhead for managing the configuration, the need to debug connections in case of network problems, and that tunneling solutions can circumvent the firewall configuration [77]. Host blocking has the disadvantage that applications connecting to the same remote servers cannot be restricted separately. Still, this approach is fairly popular on smartphones for blocking of advertisements, even though maintaining an up-to-date host list is very challenging [64]. The challenges faced here are identical to censorship scenarios: A missing host entry results in leaking sensitive information, while (unnecessary) entries might break the application's functionality and, thus, do not provide an advantage to the user.

4. Nudging Privacy on Smartphones

A wide range of different approaches to enforce privacy on smartphones are implemented in mobile operating systems (cf. Section 3). Most of these approaches have in common that they require extensive user interaction, e.g., to repeatedly react to (presented) privacy leaks and adjust the granted permissions accordingly [5, 17, 47]. These approaches to privacy enforcement are hence only successful if users are actively and continuously using them to review and configure their privacy preferences. However, often users are unaware that applications access their private data without their explicit consent, they forget about applications being installed on their device, or they are just too lazy to act on their own [17].

To overcome this dilemma, one promising attempt to trigger the user's attention is called *nudging*. Overall, nudges, i.e., the specific representation of the concept of nudging, have a vital influence on the measured success rate. An own stream of research investigates the influence of color, size, and other features that influence a nudge's appearance. For example, Choe et al. [78] evaluate nudge framing, i.e., how the information contained in a nudge manipulates the user's decision making, while Zhang et al. [79] study the presentation of nudges in the context of mobile applications.

In the context of this work, we are interested in *nudging privacy* on smartphones and hence focus on the intrusiveness, i.e., how forcefully the user's usual behavior is influenced, of nudges in the following. To this end, we differentiate between three categories, which we order by their intrusiveness. First, *hidden* nudges do not actively inform the user, i.e., they do not present prominent notifications to the user (e.g., to warn about privacy risks). Instead, the user has to explicitly access the information provided by a hidden nudge. Second, *minimized* nudges do not significantly disturb the user during application usage. Minimized nudges are presented in a non-intrusive way and should motivate the user to adjust her privacy settings. Third, *interactive* nudges force a user to interact with the nudge before she can continue with her activity. Interactive nudges represent a very intrusive form of nudging. Overall, designing privacy nudges for smartphones is a challenging task given the constraints of mobile devices such as limited screen size and restricted input possibilities.

In the following, we detail our discussion of the three categories of privacy nudges based on related research surveys. Here, it is especially challenging to rate whether a nudge is successful, i.e., helps a user in protecting her privacy. First, privacy is a very subjective aspect and there is no ground truth on whether a user should change her privacy settings in a specific situation or not [80]. Second, a more privacy-sensitive user might not require nudges to enforce privacy as she already adjusted her privacy settings in advance.

Hidden Nudges. As privacy enforcement is a highly complex topic that has to be addressed seriously, one way of nudging occurs by presenting statistics in a dedicated application that has to be explicitly accessed by the user. This way, the information can be presented in detail without disturbing the user. However, the user has to actively access the information.

A study conducted by Almuhimedi et al. [17] included an initial phase that applied hidden nudging by relying on *AppOps* (cf. Section 3.2). In a subsequent study phase, they added interactive nudging to their experiment. Their findings reveal that 22 out 23 participants accessed *AppOps* during the study phase of one week at least once without being explicitly triggered by a nudge. This result might be biased by the fact that the users were participants in a study, i.e, aware of the deployment of *AppOps* on their devices. After accessing *AppOps*, 14 of the participants significantly restricted their applications' permissions. In particular, they restricted between 6 and 49 permissions of their installed applications with focus on protecting their location and contact list. This demonstrates that statistics on privacy, e.g., application permissions, are already a valuable trigger for users to enforce privacy.

PrivacyLeaks, introduced by Balebako et al. [18], relies on hidden nudging to present additional information to the user that is not visible in the minimized nudge that is used to initially notify the user. The study which they performed with 19 participants revealed that the representation of information is an extremely vital aspect. Only 6 of the participants were able to fully understand the presented information within *PrivacyLeaks*. Nevertheless, the majority of participants rated an approach such as *PrivacyLeaks* as helpful, especially because they were unaware of the privacy leakage occurring on their smartphone before participating in the study.

Ferreira et al. [3] argue that preemptive containment of permission access, i.e., predicting an application's permission access and restricting it according to the user's attitude, is a tough challenge for inexperienced users. Thus, they allow users to specify which privacy violations *Securacy* should inform about. To this end, their approach relies on Android's permission categories. The user can opt-in to be informed if an installed application violates her privacy settings based on usage of permission-protected functionality. Due to the overall focus on network usage, their privacy results might be biased to a certain extent, as the authors expect their participants to be familiar with or at least interested in network usage and privacy.

Minimized Nudges. Minimized nudges constantly remind users about rethinking their privacy settings without interrupting their workflow. Not interrupting the workflow is a crucial aspect, as users will not deal with a potential privacy leakage if they are annoyed by the nudge. In contrast to hidden nudges, minimized

nudges are visible without accessing a particular privacy application and presented to the user in a compact form either in the notification bar or the notification drawer. Based on the information provided by a minimized nudge, the user can decide to trigger further interaction with this nudge, e.g., to receive advice on reconfiguring her privacy settings.

As part of *MockDroid*, Beresford et al. [56] initially proposed to show notification bar messages to the user while providing mocked permissions, i.e., the user is informed each time *MockDroid* grants access to faked data. Although they do not specifically refer to this as a nudge, they value their solution's intrusiveness paired with the offered opportunities for interaction.

PrivacyLeaks [18] employs minimized nudges to implement real-time privacy notifications in Android. For example, if an application leaks the user's location, *PrivacyLeaks* shows a notification in the notification bar. Balebako et al. [18] state that these notifications support the users' comprehension of privacy leakage, because they are made aware of the context in which a privacy leakage occurs. These notifications lead users to the question why the application accesses this specific information. To further study the impact of nudges, the authors performed an experiment with 10 users in which privacy notifications were announced by sounds and vibration. While this supported and emphasized the frequency of privacy leaks, users were irritated by the constant notifications. Similarly, Bal et al. [81] conduct a study with 50 participants to evaluate their implementation of minimized nudges called *Styx*. Their results indicate that *Styx* was easy to use and that it significantly increased the awareness of privacy leaks for the participants. The authors summarize that users should be only nudged if the respective system provides options to adjust permissions according to the user's preference.

To study whether users are able to map a specific permission access to the application triggering it, Thompson et al. [82] used minimized nudges to present permission access. In an experiment with 189 users, they show that only 17 % of the users were able to correctly identify the source of a permission usage request.

Interactive Nudges. While hidden and minimized nudges do not force the user to rethink her privacy settings, interactive nudges force users to interact with them. Basically, interactive nudges present a prompt to the user whenever her interaction is required to deal with a privacy leak. For example, she is immediately prompted upon a location leak and asked whether she wants to prevent this leak in the future (with different levels of detail). This approach has the advantage that the user has to deal with privacy leaking applications right away. However, a disadvantage is that the user might bluntly interact to simply get rid of the nudge.

In the second part of the *AppOps* experiment with hidden nudges, Almuhimedi et al. [17] nudged 23 participants by presenting statistics about recent permission usage. Out of the 23 participants, 22 reacted to the nudge and 14 adjusted at least one granted permission afterward. Judging these results is difficult, as most participants already constrained their applications in the first part of the experiment related to hidden nudges.

In contrast to *AppOps*, Thompson et al. [82] introduced a slightly different approach for interactive nudging. They annotate the user interface with information on the application that last had access to the permission. For example, the wallpaper configuration view informs the user about which application most recently modified the wallpaper setting. In their study, 10 out of 13 participants were not able to correctly identify the responsible applications in their experiment. Compared to other interactive nudges, this approach does not allow users to directly react to the presented information. Hence, another solution (cf. Section 3.2 and 3.3) is necessary to react to a potential privacy leak.

Based on the findings of Almuhimedi et al. [17], Liu et al. [83] created a study in which participants are nudged on a daily basis. Their nudges include additional information, such as the reason why an application most likely accessed a specific permission and the frequency with which an application accesses permissions. The participants initially had to answer five questions which were used to automatically generate a user specific profile. Eventually, the assistant gave privacy recommendations based on this feedback. The study revealed that participants accepted nearly 75 % of the proposed recommendations. Unfortunately, no study evaluates such assistance in the context of hidden or minimized nudging. Hence, it is not possible to derive which of the approaches is better suited to assist users in enforcing their privacy and whether nudges with contextual information significantly influence the user's behavior.

Another way to include contextual information into the permission granting process is to present information about permissions' purpose. Wang et al. [72] conducted a study that extends the install prompt

(cf. Section 3.1) with advertisement related toggles, i.e., distinguishing between permission access for application usage or for advertising purposes. This enables users to derive relevant information for their individual decision on privacy. However, this approach is limited to install time and does not repeatedly nudge the user. Wang et al. [72] conclude that their interactive approach shows that users are more likely to install an application if they are able to constrain the access to information the application has and are informed about access to their private data. Although this study focuses on advertisements, the gathered conclusions are likely also valid in other privacy scenarios, such as privacy enforcement on smartphones.

To summarize, we have seen three kinds of nudging approaches, which mainly differ in the way the user interacts with them. Hidden nudges have no impact on the user's workflow, however, users who do not tend to review their privacy settings on their own will most likely not encounter the nudge at all and, ultimately, not adjust their privacy enforcement. While interactive nudges force the user to deal with the granted permissions, their intrusive behavior might result in users' disapproval. Nevertheless, studies have shown that proposed recommendations alongside with contextual information are helpful, even for inexperienced users, to come up with well-founded decisions. A risk here is that users are blindly accepting recommendations without understanding or questioning their cause. Thus, minimized nudges might represent the compromise between those previous approaches. Instead of interrupting the user's workflow, minimized nudges are placed in the user interface for the user to interact with on their own terms. Studies evaluated the impact of additional notifications, such as vibrations or sounds, to underline the importance of contextual information. The results indicate that users have to be educated well, because otherwise, they are unable to correlate these notifications to privacy enforcement. Again, the risk is to annoy the user by constantly nudging her.

Concluding, we argue that there is no ideal nudging strategy as this subjective matter has to be matched to each individual users [69]. Thus, nudging performed by the operating system ideally should adjust its frequency according to the user's participation. In case this implementation turns out to be challenging, the user at least should be able to opt-out of any of the presented nudging strategies. In the following, we will connect the different privacy enforcing approaches and nudging strategies to identify open issues and future research questions.

5. Discussion and Outlook to the Future

So far, we focused on various concepts to enforce privacy on smartphones. Additionally, we had a look at nudging which attempts to solve the problem of users' ignorance or laziness in context of privacy enforcement. Overall, no ideal solution exists today, as all solutions typically involve a trade-off between the level of user manipulation and usability. Furthermore, privacy and, thus, privacy enforcement is a highly subjective topic, i.e., not a single solution is likely to work for all users [80].

In this section, we summarize the challenges we identified when it comes to privacy enforcing on smartphones to convey a deeper understanding of privacy issues especially on such a mobile platform. These issues can be roughly divided in three categories: First, *usability* of an approach for privacy enforcement is an important issue, as a solution is only mature if the user is willing to rely on it. Correspondingly, security and privacy should not suffer from good usability. Second, *users' ignorance* is a challenge, as most users do not take the effort to use any approach to privacy enforcement at all. Here, nudging is one approach to encourage users to actively exercise their right for privacy control. Third, *conceptual problems* of mobile operating systems challenge privacy on smartphones, as the related business models lead to or at least accept loss of privacy at a large scale [9].

In the following, we present these three challenges in more detail. Additionally, we attempt to highlight probable future developments based on our presented findings. Thereby, it is important to note that researchers and developers of privacy enforcement approaches can address the first two challenges, while the third challenge can only be overcome by pressure originating from a significant group of privacy conscious users and associations.

Usability. Just as in other areas, e.g., user authentication as part of access control or global email encryption [84], the trade-off between usability and security or privacy for the user has to be found when enforcing privacy on smartphones. Neglecting this important requirement will inevitably lead to denial of an approach. Hence, it is important to properly configure this trade-off.

Recently, the developer of *XPrivacy* (cf. Section 3.3) expressed his reasons why he is discontinuing active development [85]. Mainly, he identified that usability issues, such as modifying the system image for setup or the user interface's complexity, decrease user interest in his solution. Thus, users do not consider the supported enforcing capabilities as the only crucial aspect as part of their decision making. Instead of working on *XPrivacy*, the developer decided to work on a solution that utilizes the VPN interface, similar to *Haystack* (cf. Section 3.2). Hence, with this approach, the user's privacy control can be increased without root access or a modified system image. Unfortunately, this development shows what is missing on the mobile platform to refine itself in the future. First, developers cannot target as many users as they desire due to platform constraints, and therefore, they shift their focus. Second, users are not able to deploy more advanced solutions, which offer a real benefit, even though the mobile platform does not satisfy their needs. In the future, a desirable change may consist of two different aspects. On one hand, more sophisticated approaches for privacy enforcement could be integrated into mobile operating systems. On the other hand, the boundaries to securely extend the system could be lowered for both developers and users.

In Table 2, we linked usability with the level of user manipulation, as solutions without user interaction cannot properly enforce privacy as this matter is highly subjective and cannot be solved in a one-for-all manner [80]. Challenges in this area include the high number of available permissions on mobile operating systems, the level of detail the user has to face, and the intrusiveness of a solution as introduced in Sections 2 and 4. One promising approach is to group permissions so that the ordinary user only has to make a limited number of decisions. Machine learning on real user information might be feasible to improve the combination of grouped permissions and eventually reduce the number of vital permission groups per user. Lin et al. [86] already propose to group users based on their privacy preferences. Liu et al. [83] confirm this clustering and propose a privacy assistant which provides suggestions even without previous extensive user interaction. Ziegeldorf et al. [80] and Henze et al. [32] instead propose to compare user's behavior with respect to certain privacy metrics to those of her peer groups (i.e., other users with a comparable sociodemographic background or interests). Additionally, it is essential for developers to understand that there are differences in private data [10]. For example, a location-based advertisement might be acceptable as long as no movement profile can be compiled through a unique identifier. However, sensitive information stored on the device, e.g., contact lists or calendars, should be protected in any case. Nevertheless, developers could integrate a mode that allows more sophisticated users to go into more detail, thereby adapting configurability to the expertise of the individual user [15]. The user should only be challenged by privacy choices to such extend that privacy enforcement does not degenerate into agony or over-burdening.

Another major aspect that influences the usability of privacy enforcement is how frequently the user is forced to interact with the approach. This is underlined by the different nudging strategies introduced in Section 4. The question here is whether the developer is able to set privacy supporting default settings without breaking the application's functionality. This question is especially important when dealing with forged or omitted user data. *XPrivacy* (cf. Section 3.3) attempts to do so by setting sensible default values. This way the user is protected right away and no initial granting or restricting is necessary. Overall, an ideal implementation would be completely unintrusive. Unfortunately, such default settings are yet to be found. Mainly, this is because privacy is a very subjective topic, with some users being more concerned about their privacy than others. Thus, researchers have proposed a recommendation system for privacy settings [87]. Unfortunately, potential sources for such recommendations are limited. The number of applications is too high to solely rely on recommendations by experts. Furthermore, privacy leaks cannot be analyzed automatically to date. Typically, only a small fraction of users participates in crowd-sourcing platforms, potentially biasing the generated verdict. This might change in the future if the platform provider decides to extend the store's ranking system accordingly and to automatically submit made decisions. To address this issue, Taylor et al. [88] presented an information source for Android applications that incorporates privacy information from human and automated sources to generate a privacy respecting ranking. However, given that this store is just a third-party application, the user still has to rely on the Play Store and Android's permission management for application installations and permission enforcement respectively. From a different perspective, Quay-de la Vallee et al. [89] presented *SecuRank*, a framework which proposes the user alternatives to permission-hungry applications. However, this list might contain incorrect results as *SecuRank* simply groups the applications by their store's description. Similarly, Moussa et al. [90]

implemented *ACCUSE* which ranks applications according to a risk level they calculated based on requested permissions. In addition, they graphically visualize the impact of permissions according to their individual risk levels.

To conclude, the frequency of asking the user to adjust her settings or to make a decision is vital to the success of privacy enforcement on smartphones. On one hand, a detailed configuration should support users in controlling their privacy. On the other hand, repeated notifications might discourage users and result in laziness or ignorance. In the future, automated approaches should be able to reduce the number of decisions users have to make to shift the users' focus to a limited number important and detailed configuration options.

Users' Ignorance. Surveys revealed that ordinary users are not aware of the privacy leaks occurring through their mobile device, i.e., on their own, they usually do not reflect on privacy [17, 18, 82]. Therefore, it is of utmost importance to encourage users to enforce their privacy. An acceptable solution seems to be nudging which does not annoy the user by interrupting her workflow [82]. Eventually, the (somewhat) privacy-sensitive user will be reminded, often enough and at a suitable time, to enforce her right to privacy. Unfortunately, ignorance by one user causes a chain reaction as the privacy of a single user is influenced by related users, mostly because parts of sensitive data, such as contact lists or emails, overlap among a group of users. Hence, one user's decisions consequently have a huge impact on other users' privacy.

For example, if the contact list of one user leaks to a cloud service, the privacy of all users on that contact list is impaired. This stresses the importance of a widespread solution for supporting users in the enforcement of their privacy. However, users' ignorance has various origins. Some of those might be impossible to tackle, e.g., a user not concerned about privacy at all. Still, other origins can be easily approached. For example, users overwhelmed by applications or afraid of breaking their system should be supported accordingly as mentioned in Section 4. In our opinion, providing meaningful statistics significantly improves the user's awareness of privacy problems and tutorials are a simple way to familiarize users with privacy-protecting approaches. Finally, the operating system should support privacy enforcement to a larger extend in the future, because application crashes, e.g., due to missing permissions, annoy users and discourage them from protecting their privacy [18].

Apart from restricting permissions afterward, the issue of finding the ideal way for permission granting still exists with regard to upholding the users' privacy. The core question here is whether it is better to grant permissions right at the beginning and possibly restrict them later on, or whether the user should constrain the application's permissions right from the beginning. Here, the amount of required interaction is a key issue, i.e., how severely the user's workflow is interrupted. Consequentially, the question how often to nudge is also a vital challenge, as nudges remind the user to adjust her privacy settings and depending on their intrusiveness, the user's workflow is significantly influenced. Additionally, a privacy-sensitive user might require far fewer nudges than an inexperienced uneducated user as the initial configuration of an experienced user is already reasonable with respect to privacy. However, we believe that users' ignorance will decrease if nudging becomes more accepted by the majority of users to trigger privacy enforcement. Hence, it only seems to be a matter of large-scale deployment.

Conceptual Problems. In a way, mobile operating systems can be rated as malicious, because, currently, they do not include enough possibilities to enforce privacy [6]. Most options, such as application isolation and permission-based access control, that are widely used, are only included because security aspects mandate them [13]. As long as system developers benefit from sensitive data and personalized ads, in particular, the situation is doomed to fail. In Android, mainly developed by *Google*, many third-party developers rely on *Google Ads* to monetize their applications [10]. While this offers a business model for developers, it jeopardizes the privacy of users. Interestingly, it remains to be seen whether privacy supporting approaches have a negative influence on the revenue model of application developers if widely applied. Egelman et al. [91] might have shown a way out of the dilemma between privacy and monetization by pointing out that users are willing to pay for privacy if necessary and if they are given a choice to do so. Leontiadis et al. [92] proposed a framework that automatically adjusts the amount of shared private information, according to the advertisement revenue generated by the developer, to minimize the privacy leakage.

Nevertheless, a not so drastic but still realistic approach is to decouple advertising libraries from applications. This would allow a more open privacy analysis, i.e., which permissions are actually required by the

core application and which are used for advertising. Such an analysis could reveal what kind of user profiles the application is able to generate. For Android, implementations of this idea, e.g, *AdSplit* [10] and *AdDroid* [9], have been proposed.

We observe an increasing trend of smartphone applications to communicate with cloud services, most notably to overcome inherently limited resources of smartphones with respect to computational power, storage space, and energy [3, 5]. As a result, the privacy risks of applications having access to private information stored, processed, and sensed by smartphones further amplify: Private information now has an increased risk of being exposed to third-parties, ranging from (indirectly) utilized cloud providers over government agencies to hackers [93]. These risks mainly result from the inherent centrality, non-transparency, technical complexity, and opaque legislation of cloud computing [93]. These increased privacy risks lead to users perceiving a loss of control over the sensitive information on their smartphone when it is forwarded to the cloud [93, 94]. Thus, it is important to consider that not only applications can (mis)use granted permissions, but that all cloud services contacted by an application can amplify users' privacy risks, motivating the need to make users aware of these risks and to develop appropriate countermeasures [32].

Another conceptual problem is that mobile operating systems with install prompts (up to Android 6 and Windows Phone) do not offer the possibility to restrict permissions once they have been granted [6]. Solutions adding this kind of functionality do not always add a benefit to the user, because applications tend to crash with missing permissions [6]. This phenomenon results from the fact that developers are not forced to properly handle these situations and they do not expect that permissions might be revoked. With the introduction of runtime permissions and the option of toggling permissions in Android 6 [20], it is likely that this situation will improve at least for actively developed Android applications. To improve the overall acceptance of user privacy, Balebako et al. [95] propose to nudge *developers* to protect their users' privacy. To support developers, *P-Lint* [96] analyzes Android applications with respect to improper permission usage, which can result in user confusion in the best case, or in severe security and privacy issues in the worst case. A tool, presented by Vidas et al. [97], also supports developers to implement their applications with as few permissions as possible. This is an important milestone as it shows that privacy control can also be triggered by the software development industry.

As the developers of mobile operating systems currently have no incentive of adjusting their revenue model and, thus, their principle of treating users' privacy, it might be necessary to develop corresponding incentives. For example, the government could pass laws that would force developers to respect or even improve users' privacy. Nevertheless, if users would show an increasing demand or interest into privacy control (e.g., because their awareness of privacy increases), developers would already be forced to adjust their behavior to satisfy the market. We believe that this interest might be triggered at some point in the future as we have seen with the general privacy debate following the NSA revelations. Unfortunately, most (inexperienced) users might not be able to address their concerns, if any, to appropriate places. Hence, users need to be educated and informed about their options and the difference they can make.

6. Conclusion

Privacy enforcement is a tremendous challenge for smartphone users as of today. The reasons for this are diverse: On the one hand, users are not experienced enough to understand the correlation between granted permissions to applications and privacy leaks (cf. Section 4). On the other hand, solutions to deal with these problems are not ideal, i.e., they often offer a bad usability or an insufficient level of user manipulation (cf. Section 3). In this paper, we compared existing solutions and acknowledged their potential, while noting that the ideal candidate is yet to be found. Some of the presented approaches, such as install time and runtime prompting, are already implemented in current mobile operating systems while others simply represent interesting research approaches.

A significant problem for privacy enforcement is users' laziness. As a result, privacy nudging attempts to recurringly remind users of their right to privacy which most users do not exercise on their own. The intrusiveness of nudges directly correlates with their usability. An annoying solution is bound to fail, while rare nudging can waste a lot of potential for privacy enforcement. Overall, there are advantages and

disadvantages for all nudging concepts. Nevertheless, studies have shown that most users rate nudges as helpful and supporting.

In the future, privacy enforcement on smartphones will remain an important topic, hence, more approaches and research will emerge. Especially the introduction of runtime permissions in Android is a change towards a more privacy restrictive implementation as the operating system grants the user more possibilities for enforcing her privacy. This change, in the most commonly used mobile operating system, will clearly shift focus towards more privacy enforcement. It remains to be seen whether more fundamentally changes (cf. Section 5) will follow. So far, no ideal solution for all stakeholders, such as users, developers, and companies, has been presented. Most likely, users have to take care of themselves, because, ultimately, privacy is a valuable good for them which has to be treated accordingly. For this reason, inexperienced users have to be supported so that they are able to decide for themselves, whether they want their privacy to be invaded or enforced.

Acknowledgments

This work has in parts been funded by the German Federal Ministry of Education and Research (BMBF) under project funding reference number 16KIS0351 (TRINICS). The responsibility for the content of this publication lies with the authors.

References

[1] IDC Research Inc. , Smartphone OS Market Share, 2017 Q1, https://www.idc.com/prodserv/smartphone-os-market-share.jsp (2017 (accessed July 14, 2017)).

[2] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, C. Cowan, User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems, in: Proceedings of the 33rd IEEE Symposium on Security and Privacy (SP '12), IEEE Computer Society, 2012, pp. 224–238. doi:10.1109/SP.2012.24.

[3] D. Ferreira, V. Kostakos, A. R. Beresford, J. Lindqvist, A. K. Dey, Securacy: An Empirical Investigation of Android Applications' Network Usage, Privacy and Security, in: Proceedings of the 8th Conference on Security & Privacy in Wireless and Mobile Networks (WiSec '15), ACM, 2015, pp. 11:1–11:11. doi:10.1145/2766498.2766506.

[4] M. Henze, L. Hermerschmidt, D. Kerpen, R. Häußling, B. Rumpe, K. Wehrle, A Comprehensive Approach to Privacy in the Cloud-based Internet of Things, Future Generation Computer Systems 56 (2016) 701–718. doi:10.1016/j.future.2015.09.016.

[5] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI '10), USENIX Association, 2010, pp. 393–407.

[6] N. Asokan, L. Davi, A. Dmitrienko, S. Heuser, K. Kostiainen, E. Reshetova, A.-R. Sadeghi, Mobile Platform Security, Vol. 4 of Synthesis lectures on Information Security, Privacy, and Trust, Morgan & Claypool, 2013. doi:10.2200/S00555ED1V01Y201312SPT009.

[7] WhatsApp Inc. , Why does WhatsApp use my phone number and my address book?, https://www.whatsapp.com/faq/en/general/20971813 (2016 (accessed November 11, 2016)).

[8] C. Gibler, J. Crussell, J. Erickson, H. Chen, AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale, in: Proceedings of the 5th International Conference on Trust and Trustworthy Computing (TRUST '12), Springer-Verlag, 2012, pp. 291–307. doi:10.1007/978-3-642-30921-2_17.

[9] P. Pearce, A. P. Felt, G. Nunez, D. Wagner, AdDroid: Privilege Separation for Applications and Advertisers in Android, in: Proceedings of the 7th Symposium on Information, Computer and Communications Security (ASIA CCS '12), ACM, 2012, pp. 71–72. doi:10.1145/2414456.2414498.

[10] S. Shekhar, M. Dietz, D. S. Wallach, AdSplit: Separating Smartphone Advertising from Applications, in: Proceedings of the 21st USENIX Conference on Security Symposium (Security '12), USENIX Association, 2012, pp. 28–28.

[11] B. Young, Announcement: Amazon Underground Actually Free Program, https://developer.amazon.com/blogs/appstore/post/cbadeae1-990d-4d52-bef5-ea61f6114b94/announcement-amazon-actually-free-program (2017 (accessed May 23, 2017)).

[12] Amazon.com Inc. , Amazon Underground, https://developer.amazon.com/public/solutions/underground (2016 (accessed November 13, 2016)).

[13] N. Elenkov, Android Security Internals: An In-depth Guide to Android's Security Architecture, 1st Edition, No Starch Press, 2014.

[14] C. Spensky, J. Stewart, A. Yerukhimovich, R. Shay, A. Trachtenberg, R. Housley, R. K. Cunningham, SoK: Privacy on Mobile Devices – It's Complicated, in: Proceedings of the 16th Symposium on Privacy Enhancing Technologies (PETS '16), De Gruyter, 2016, pp. 96–116. doi:10.1515/popets-2016-0018.

[15] M. Henze, L. Hermerschmidt, D. Kerpen, R. Häußling, B. Rumpe, K. Wehrle, User-driven Privacy Enforcement for Cloud-based Services in the Internet of Things, in: Proceedings of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud '14), IEEE Computer Society, 2014, pp. 191–196. doi:10.1109/FiCloud.2014.38.

[16] M. Nauman, S. Khan, X. Zhang, Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints, in: Proceedings of the 5th Symposium on Information, Computer and Communications Security (ASIA CCS '10), ACM, 2010, pp. 328–332. doi:10.1145/1755688.1755732.

[17] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, Y. Agarwal, Your Location has been Shared 5,398 Times!: A Field Study on Mobile App Privacy Nudging, in: Proceedings of the 33rd ACM Conference on Human Factors in Computing Systems (CHI '15), ACM, 2015, pp. 787–796. doi:10.1145/2702123.2702210.

[18] R. Balebako, J. Jung, W. Lu, L. F. Cranor, C. Nguyen, "Little Brothers Watching You": Raising Awareness of Data Leaks on Smartphones, in: Proceedings of the 9th USENIX Symposium on Usable Privacy and Security (SOUPS '13), USENIX Association, 2013, pp. 12:1–12:11. doi:10.1145/2501604.2501616.

[19] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, D. Wagner, How to Ask for Permission, in: Proceedings of the 7th USENIX Conference on Hot Topics in Security (HotSec '12), USENIX Association, 2012, pp. 7–7.

[20] Google Inc. , Android Developers, https://developer.android.com/ (2016 (accessed November 20, 2016)).

[21] Google Inc. , Android Developers - Permissions, https://developer.android.com/guide/topics/security/permissions.html (2016 (accessed November 20, 2016)).

[22] Y. Fratantonio, C. Qian, S. Chung, W. Lee, ARTist: The Android Runtime Instrumentation and Security Toolkit, in: Proceedings of the 38th IEEE Symposium on Security and Privacy (SP '17), IEEE Computer Society, 2017, pp. 1041–1057. doi:10.1109/SP.2017.39.

[23] Apple Inc. , iOS Developer Library - Supported Capabilities, https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/SupportedCapabilities/SupportedCapabilities.html (2016 (accessed November 20, 2016)).

[24] Microsoft, Windows Dev Center - App capabilities, https://msdn.microsoft.com/en-us/library/windows/apps/jj206936 (2016 (accessed November 20, 2016)).

[25] J. Sellwood, J. Crampton, Sleeping Android: The Danger of Dormant Permissions, in: Proceedings of the 3rd ACM Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM '13), ACM, 2013, pp. 55–66. doi:10.1145/2516760.2516774.

[26] L. Xing, X. Pan, R. Wang, K. Yuan, X. Wang, Upgrading Your Android, Elevating My Malware: Privilege Escalation Through Mobile OS Updating, in: Proceedings of the 35th IEEE Symposium on Security and Privacy (SP '14), IEEE Computer Society, 2014, pp. 393–408. doi:10.1109/SP.2014.32.

[27] W. Xu, F. Zhang, S. Zhu, Permlyzer: Analyzing permission usage in Android applications, in: Proceedings of the 24th IEEE Symposium on Software Reliability Engineering (ISSRE '13), IEEE Computer Society, 2013, pp. 400–410. doi:10.1109/ISSRE.2013.6698893.

[28] S. Rosen, Z. Qian, Z. M. Mao, AppProfiler: A Flexible Method of Exposing Privacy-related Behavior in Android Applications to End Users, in: Proceedings of the 3rd Conference on Data and Application Security and Privacy (CODASPY '13), ACM, 2013, pp. 221–232. doi:10.1145/2435349.2435380.

[29] J. Ren, A. Rao, M. Lindorfer, A. Legout, D. Choffnes, ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic, in: Proceeding of the 14th Conference on Mobile Systems, Applications, and Services (MobiSys '16), ACM, 2016, pp. 361–374. doi:10.1145/2906388.2906392.

[30] E. Reshetova, F. Bonazzi, N. Asokan, SELint: An SEAndroid Policy Analysis Tool, in: Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP '17), SciTePress, 2017, pp. 47–58. doi:10.5220/0006126600470058.

[31] P. K. Tiwari, U. Singh, Android Users Security via Permission Based Analysis, in: Proceedings of the 3rd International Symposium on Security in Computing and Communications (SSCC '15), Springer-Verlag, 2015, pp. 496–505. doi:10.1007/978-3-319-22915-7_45.

[32] M. Henze, D. Kerpen, J. Hiller, M. Eggert, D. Hellmanns, E. Mühmer, O. Renuli, H. Maier, C. Stüble, R. Häußling, K. Wehrle, Towards Transparent Information on Individual Cloud Service Usage, in: Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom '16), IEEE Computer Society, 2016, pp. 366–370. doi:10.1109/CloudCom.2016.0064.

[33] H. Harkous, R. Rahman, B. Karlas, K. Aberer, The Curious Case of the PDF Converter that Likes Mozart: Dissecting and Mitigating the Privacy Risk of Personal Cloud Apps, in: Proceedings of the 16th Symposium on Privacy Enhancing Technologies (PETS '16), De Gruyter, 2016, pp. 123–143. doi:10.1515/popets-2016-0032.

[34] M. Henze, M. P. Sanford, O. Hohlfeld, Veiled in Clouds? Assessing the Prevalence of Cloud Computing in the Email Landscape, in: 2017 IEEE/IFIP Network Traffic Measurement and Analysis Conference (TMA), IEEE/IFIP, 2017.

[35] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, A. Markopoulou, AntMonitor: A System for Monitoring from Mobile Devices, in: Proceedings of the 1st SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data (C2B(1)D '15), ACM, 2015, pp. 15–20. doi:10.1145/2787394.2787396.

[36] Y. Rahulamathavan, V. Moonsamy, L. Batten, S. Shunliang, M. Rajarajan, An Analysis of Tracking Settings in Blackberry 10 and Windows Phone 8 Smartphones, in: Proceeding of the 19th Australasian Conference on Information Security and Privacy (ACISP '14), Springer-Verlag, 2014, pp. 430–437. doi:10.1007/978-3-319-08344-5_29.

[37] Chainfire, Root without modifying /system, http://forum.xda-developers.com/showpost.php?p=63197935&postcount=2 (2015 (accessed November 27, 2016)).

[38] topjohnwu, Xposed - Universal Systemless Compatibility, http://forum.xda-developers.com/xposed/unofficial-systemless-xposed-t3388268 (2016 (accessed November 27, 2016)).

[39] M. Egele, C. Kruegel, E. Kirda, G. Vigna, PiOS: Detecting Privacy Leaks in iOS Applications, in: Proceedings of the 18th ISOC Network and Distributed System Security Symposium (NDSS '11), ISOC, 2011, pp. 177–183.
[40] F-Secure Corporation, Play Store - F-Secure App Permissions, https://play.google.com/store/apps/details?id=com.fsecure.app.permissions.privacy (2015 (accessed November 11, 2016)).
[41] C. Qian, X. Luo, Y. Shao, A. T. S. Chan, On Tracking Information Flows through JNI in Android Applications, in: Proceedings of the 44th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '14), IEEE Computer Society, 2014, pp. 180–191. doi:10.1109/DSN.2014.30.
[42] M. Sun, T. Wei, J. C. Lui, TaintART: A Practical Multi-level Information-Flow Tracking System for Android RunTime, in: Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS '16), ACM, 2016, pp. 331–342. doi:10.1145/2976749.2978343.
[43] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, V. Paxson, Haystack: In Situ Mobile Traffic Analysis in User Space, in: arXiv preprint arXiv:1510.01419, arXiv, 2015, pp. 1–13.
[44] V. Rastogi, Z. Qu, J. McClurg, Y. Cao, Y. Chen, Uranine: Real-time Privacy Leakage Monitoring without System Modification for Android, in: Proceedings of the 11th Conference on Security and Privacy in Communication Networks (SecureComm '15), ACM, 2015, pp. 256–276. doi:10.1007/978-3-319-28865-9_14.
[45] B. Rashidi, C. Fung, T. Vu, Android fine-grained permission control system with real-time expert recommendations, in: Pervasive and Mobile Computing, Elsevier, 2016, pp. 1–31. doi:10.1016/j.pmcj.2016.04.013.
[46] Lamian, Play Store - LBE (Root), https://play.google.com/store/apps/details?id=com.lbe.security (2015 (accessed November 17, 2016)).
[47] M. Backes, S. Gerling, C. Hammer, M. Maffei, P. von Styp-Rekowsky, AppGuard: Enforcing User Requirements on Android Apps, in: Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13), Springer-Verlag, 2013, pp. 543–548. doi:10.1007/978-3-642-36742-7_39.
[48] M. Backes, S. Bugiel, O. Schranz, P. von Styp-Rekowsky, S. Weisgerber, ARTist: The Android Runtime Instrumentation and Security Toolkit, in: Proceedings of the 2nd European Symposium on Security and Privacy (Euro S&P '17), IEEE Computer Society, 2017, pp. 481–495. doi:10.1109/EuroSP.2017.43.
[49] M. Zhang, H. Yin, Efficient, Context-aware Privacy Leakage Confinement for Android Applications Without Firmware Modding, in: Proceedings of the 9th Symposium on Information, Computer and Communications Security (ASIA CCS '14), ACM, 2014, pp. 259–270. doi:10.1145/2590296.2590312.
[50] S. Biswas, W. Haipeng, J. Rashid, Android Permissions Management at App Installing, in: International Journal of Security and Its Applications, Vol. 10, SERSC, 2016, pp. 223–232.
[51] F. Roesner, T. Kohno, Securing Embedded User Interfaces: Android and Beyond, in: Proceedings of the 22nd USENIX Conference on Security (SEC '13), USENIX Association, 2013, pp. 97–112.
[52] T. Ringer, D. Grossman, F. Roesner, AUDACIOUS: User-Driven Access Control with Unmodified Operating Systems, in: Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS '16), ACM, 2016, pp. 204–216. doi:10.1145/2976749.2978344.
[53] P. Hornyack, S. Han, J. Jung, S. Schechter, D. Wetherall, These Aren'T the Droids You'Re Looking for: Retrofitting Android to Protect Data from Imperious Applications, in: Proceedings of the 18th Conference on Computer and Communications Security (CCS '11), ACM, 2011, pp. 639–652. doi:10.1145/2046707.2046780.
[54] M66B, GitHub - XPrivacy, https://github.com/M66B/XPrivacy (2013 (accessed December 5, 2016)).
[55] rovo89, GitHub - Xposed, https://github.com/rovo89/Xposed (2012 (accessed December 5, 2016)).
[56] A. R. Beresford, A. Rice, N. Skehin, R. Sohan, MockDroid: Trading Privacy for Application Functionality on Smartphones, in: Proceedings of the 12th Workshop on Mobile Computing Systems & Applications (HotMobile '11), ACM, 2011, pp. 49–54. doi:10.1145/2184489.2184500.
[57] Y. Zhou, X. Zhang, X. Jiang, V. W. Freeh, Taming Information-stealing Smartphone Applications (on Android), in: Proceedings of the 4th International Conference on Trust and Trustworthy Computing (TRUST '11), Springer-Verlag, 2011, pp. 93–107.
[58] Y. Agarwal, M. Hall, ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing, in: Proceeding of the 11th Conference on Mobile Systems, Applications, and Services (MobiSys '13), ACM, 2013, pp. 97–110. doi:10.1145/2462456.2464460.
[59] Carnegie Mellon University (CMU), Protect My Privacy, http://www.android.protectmyprivacy.org/ (2015 (accessed December 13, 2016)).
[60] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, N. Asokan, ConXsense: Automated Context Classification for Context-aware Access Control, in: Proceedings of the 9th Symposium on Information, Computer and Communications Security (ASIA CCS '14), ACM, 2014, pp. 293–304. doi:10.1145/2590296.2590337.
[61] S. Bugiel, S. Heuser, A.-R. Sadeghi, Flexible and Fine-Grained Mandatory Access Control on Android for Diverse Security and Privacy Policies, in: Proceedings of the 22nd USENIX Conference on Security (SEC '13), USENIX Association, 2013, pp. 131–146.
[62] L. Tsai, P. Wijesekera, J. Reardon, I. Reyes, J.-W. Chen, N. Good, S. Egelman, D. Wagner, Turtle Guard: Helping Android Users Apply Contextual Privacy Preferences, in: Proceedings of the 13th USENIX Symposium on Usable Privacy and Security (SOUPS '17), USENIX Association, 2017, pp. 145–162.
[63] ukanth, GitHub - AFWall+, https://github.com/ukanth/afwall (2012 (accessed December 9, 2016)).
[64] Free-Software-for-Android, GitHub - AdAway, https://github.com/Free-Software-for-Android/AdAway (2011 (accessed December 9, 2016)).
[65] Katriina_M, F-Secure App Permissions will be discontinued on December 4, 2015, https://community.f-secure.com/t5/F-Secure-SAFE/F-Secure-App-Permissions-will-be/ta-p/77859 (2017 (accessed May 28, 2017)).

[66] Intel Labs, Penn State University, Duke University, Realtime Privacy Monitoring on Smartphones, http://appanalysis.org/download.html (2013 (accessed December 3, 2016)).
[67] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, X. S. Wang, AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection, in: Proceedings of the 20th Conference on Computer and Communications Security (CCS '13), ACM, 2013, pp. 1043–1054. doi:10.1145/2508859.2516676.
[68] International Computer Science Institute, UC Berkeley, Play Store - Lumen Privacy Monitor, https://play.google.com/store/apps/details?id=edu.berkeley.icsi.haystack (2015 (accessed May 30, 2017)).
[69] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner, Android Permissions: User Attention, Comprehension, and Behavior, in: Proceedings of the 8th Symposium on Usable Privacy and Security (SOUPS '12), ACM, 2012, pp. 1–14. doi:10.1145/2335356.2335360.
[70] S. Galand, Play Store - AppOps, https://play.google.com/store/apps/details?id=fr.slvn.appops (2014 (accessed November 17, 2016)).
[71] xdadrm, GitHub - LBE Translation, https://github.com/xdadrm/lbe_translation_xposed_en (2013 (accessed November 17, 2016)).
[72] N. Wang, B. Zhang, B. Liu, H. Jin, Investigating Effects of Control and Ads Awareness on Android Users' Privacy Behaviors and Perceptions, in: Proceedings of the 17th Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '15), ACM, 2015, pp. 373–382. doi:10.1145/2785830.2785845.
[73] T. Baarslag, A. T. Alan, R. C. Gomer, I. Liccardi, H. Marreiros, E. H. Gerding, M. Schraefel, Negotiation as an Interaction Mechanism for Deciding App Permissions, in: Proceedings of the 25th Conference Extended Abstracts on Human Factors in Computing Systems (CHI '16), ACM, 2016, pp. 2012–2019. doi:10.1145/2851581.2892340.
[74] rovo89, GitHub - Xposed: when the Android 7.0 Nougat would be supported by xposed?, https://github.com/rovo89/Xposed/issues/225#issuecomment-314017010 (2017 (accessed July 12, 2017)).
[75] S. Nawaz, C. Mascolo, Mining Users' Significant Driving Routes with Low-power Sensors, in: Proceedings of the 12th Conference on Embedded Network Sensor Systems (SenSys '14), ACM, 2015, pp. 236–250. doi:10.1145/2668332.2668348.
[76] Yllier, Firewall iP, http://yllier.net/FirewalliP/FiPDepiction.html (2015 (accessed December 9, 2016)).
[77] T. Gray, Firewalls: Friend or Foe?, EDUCAUSE review (2003) 60–61.
[78] E. K. Choe, J. Jung, B. Lee, K. Fisher, Nudging People Away from Privacy-Invasive Mobile Apps through Visual Framing, in: Proceedings of the 29th 14th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT '13), Springer-Verlag, 2013, pp. 74–91. doi:10.1007/978-3-642-40477-1_5.
[79] B. Zhang, H. Xu, Privacy Nudges for Mobile Applications: Effects on the Creepiness Emotion and Privacy Attitudes, in: Proceedings of the 19th Conference on Computer-Supported Cooperative Work & Social Computing (CSCW '16), ACM, 2016, pp. 1676–1690. doi:10.1145/2818048.2820073.
[80] J. H. Ziegeldorf, M. Henze, R. Hummen, K. Wehrle, Comparison-based Privacy: Nudging Privacy in Social Media (Position Paper), in: The 10th DPM International Workshop on Data Privacy Management (DPM '15), Springer, 2015, pp. 226–234. doi:10.1007/978-3-319-29883-2_15.
[81] G. Bal, K. Rannenberg, J. Hong, Styx: Design and Evaluation of a New Privacy Risk Communication Method for Smartphones, in: Proceedings of the 29th IFIP Information Security & Privacy Conference (IFIP SEC '14), Springer-Verlag, 2014, pp. 113–126. doi:10.1007/978-3-642-55415-5_10.
[82] C. Thompson, M. Johnson, S. Egelman, D. Wagner, J. King, When It's Better to Ask Forgiveness Than Get Permission: Attribution Mechanisms for Smartphone Resources, in: Proceedings of the 9th Symposium on Usable Privacy and Security (SOUPS '13), ACM, 2013, pp. 1–14. doi:10.1145/2501604.2501605.
[83] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. Zhang, N. Sadeh, A. Acquisti, Y. Agarwal, Follow My Recommendations: A Personalized Privacy Assistant for Mobile App Permissions, in: Proceedings of the 12th USENIX Symposium on Usable Privacy and Security (SOUPS '16), USENIX Association, 2016, pp. 27–41.
[84] B. D. Payne, W. K. Edwards, A Brief Introduction to Usable Security, IEEE Internet Computing 12 (2008) 13–21. doi:10.1109/MIC.2008.50.
[85] M66B, XPrivacy - The ultimate, yet easy to use, privacy manager, http://forum.xda-developers.com/showpost.php?p=68128626&postcount=17325 (2016 (accessed December 17, 2016)).
[86] J. Lin, B. Liu, N. Sadeh, J. I. Hong, Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings, in: Proceedings of the 10th USENIX Symposium on Usable Privacy and Security (SOUPS '14), USENIX Association, 2014, pp. 199–212.
[87] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, J. Zhang, Expectation and Purpose: Understanding Users' Mental Models of Mobile App Privacy Through Crowdsourcing, in: Proceedings of the 14th Conference on Ubiquitous Computing (UbiComp '12), ACM, 2012, pp. 501–510. doi:10.1145/2370216.2370290.
[88] V. F. Taylor, I. Martinovic, SecuRank: Starving Permission-Hungry Apps Using Contextual Permission Analysis, in: Proceedings of the 6th Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM '16), ACM, 2016, pp. 43–52. doi:10.1145/2994459.2994474.
[89] H. Quay-de la Vallee, P. Selby, S. Krishnamurthi, On a (Per)Mission: Building Privacy Into the App Marketplace, in: Proceedings of the 6th Workshop on Security and Privacy in Smartphones & Mobile Devices (SPSM '16), ACM, 2016, pp. 63–72. doi:10.1145/2994459.2994466.
[90] M. Moussa, M. Di Penta, G. Antoniol, G. Beltrame, ACCUSE: Helping Users to Minimize Android App Privacy Concerns, in: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17), IEEE Computer Society, 2017, pp. 144–148. doi:10.1109/MOBILESoft.2017.22.
[91] S. Egelman, A. P. Felt, D. Wagner, Choice Architecture and Smartphone Privacy: There's a Price for That, in: The Economics of Information Security and Privacy, Springer-Verlag, 2013, pp. 211–236. doi:10.1007/978-3-642-39498-0_10.

[92] I. Leontiadis, C. Efstratiou, M. Picone, C. Mascolo, Choice Architecture and Smartphone Privacy: There's a Price for That, in: Proceedings of the 12th Workshop on Mobile Computing Systems & Applications (HotMobile '12), ACM, 2012, pp. 7–12. doi:10.1145/2162081.2162084.

[93] M. Henze, J. Hiller, O. Hohlfeld, K. Wehrle, Moving Privacy-Sensitive Services from Public Clouds to Decentralized Private Clouds, in: 2016 IEEE International Conference on Cloud Engineering Workshops, IEEE, 2016. doi:10.1109/IC2EW.2016.24.

[94] I. Ion, N. Sachdeva, P. Kumaraguru, S. Čapkun, Home is Safer Than the Cloud!: Privacy Concerns for Consumer Cloud Storage, in: Proceedings of the Seventh Symposium on Usable Privacy and Security (SOUPS '11), ACM, 2011. doi:10.1145/2078827.2078845.

[95] R. Balebako, L. Cranor, Improving App Privacy: Nudging App Developers to Protect User Privacy, in: Proceedings of the 35th IEEE Symposium on Security and Privacy (SP '14), IEEE Computer Society, 2014, pp. 55–58. doi:10.1109/MSP.2014.70.

[96] C. Dennis, D. E. Krutz, M. W. Mkaouer, Improving App Privacy: Nudging App Developers to Protect User Privacy, in: Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17), IEEE Computer Society, 2017, pp. 219–220. doi:10.1109/MOBILESoft.2017.24.

[97] T. Vidas, N. Christin, L. Cranor, Curbing Android Permission Creep, in: Proceedings of the 5th Conference on Web 2.0 Security and Privacy 2011 (W2SP '11), IEEE Computer Society, 2011, pp. 1–5.