SmarPer: Context-Aware and Automatic Runtime-Permissions for Mobile Devices

Katarzyna Olejnik, Italo Dacosta, Joana Soares Machado, Kévin Huguenin, Mohammad Emtiyaz Khan, Jean-Pierre Hubaux

SmarPer: Context-Aware and Automatic Runtime-Permissions for Mobile Devices

Katarzyna Olejnik[*,1], Italo Dacosta[†], Joana Soares Machado[†], Kévin Huguenin[‡,1], Mohammad Emtiyaz Khan[§,1], Jean-Pierre Hubaux[†]
[*]Raytheon BBN Technologies, Cambridge, MA, USA
[†]School of Computer and Communication Sciences (IC), EPFL, Lausanne, Switzerland
[‡]Faculty of Business and Economics (HEC), UNIL, Lausanne, Switzerland
[§]Center for Advanced Intelligence Project (AIP), RIKEN, Tokyo, Japan

***Abstract*—Permission systems are the main defense that mobile platforms, such as Android and iOS, offer to users to protect their private data from prying apps. However, due to the tension between usability and control, such systems have several limitations that often force users to overshare sensitive data. We address some of these limitations with *SmarPer*, an advanced permission mechanism for Android. To address the rigidity of current permission systems and their poor matching of users' privacy preferences, *SmarPer* relies on contextual information and machine learning methods to predict permission decisions at runtime. Note that the goal of *SmarPer* is to *mimic* the users' decisions, not to make privacy-preserving decisions per se. Using our *SmarPer* implementation, we collected 8,521 runtime permission decisions from 41 participants in real conditions. With this unique data set, we show that using an efficient Bayesian linear regression model results in a mean correct classification rate of 80% (±3%). This represents a mean relative reduction of approximately 50% in the number of incorrect decisions when compared with a user-defined static permission policy, i.e., the model used in current permission systems. *SmarPer* also focuses on the suboptimal trade-off between privacy and utility; instead of only "allow" or "deny" type of decisions, *SmarPer* also offers an "obfuscate" option where users can still obtain utility by revealing partial information to apps. We implemented obfuscation techniques in *SmarPer* for different data types and evaluated them during our data collection campaign. Our results show that 73% of the participants found obfuscation useful and it accounted for almost a third of the total number of decisions. In short, we are the first to show, using a large dataset of real in situ permission decisions, that it is possible to learn users' unique decision patterns at runtime using contextual information while supporting data obfuscation; this is an important step towards automating the management of permissions in smartphones.**

I. Introduction

Smartphones can be considered the most personal computing devices we have today, due to their popularity and the increasing amount of personal information they collect. To control third-party apps' access to this sensitive information, mobile platforms such as Android and iOS rely on a permission system where users can allow or deny apps' permission requests. In general, users define an access control policy that is enforced by the mobile OS at runtime.

Unfortunately, due to the trade-off between usability (i.e., permission management) and the level of control offered (i.e., granularity of permissions), current mobile permission systems have several limitations. For instance, users' permission decisions represent a *static policy*, i.e., once a permission decision is made, it will not change without user intervention (Android 6+ and iOS). This approach assumes that permission decisions are not context-dependent and rarely change over time. Yet, researchers have shown evidence of the contrary [1], [2]. For example, a user might be willing to grant an app access to her location if she is using it, but she might be reluctant to do so if the app is in the background. Our user survey (Section V) confirms this idea: Only 19% of the participants stated that context is not important to them. The results of our study also show that users' decisions are not static—for similar permission requests, many participants changed their decision at least once.

To support context-aware permission policies, a simple approach is to prompt users at runtime to make a decision. In this way, users will have more contextual information and a better understanding of the purpose of the request [2]. Android 6+ and iOS support permission decisions at runtime, but only the first time an app requests a permission. Hence, the resulting policy only captures a single user's context. CyanogenMod [3] and permission tools such as XPrivacy [4] and LBE Privacy Guard [5] offer users with an "always-ask" option to indicate what requests should be always prompted at runtime. This approach enables a better matching of users' privacy preferences, but it requires a significant effort from the user. For example, a single app can make tens to hundreds of sensitive requests per day, even if the user is not interacting with it [2]. As users have on average close to 95 apps [6] and each app requires around 5 permissions [7], it is clear that runtime decisions can overwhelm users or cause habituation to prompts. Hence, to support context-aware permission policies, advanced mechanisms are needed to help users with the overhead introduced by runtime permissions.

Another important limitation of current permission systems is their sub-optimal trade-off between privacy and utility, as users can only allow (i.e., no privacy) or deny (i.e., no utility) access to their private information. As a result, to benefit from apps' functionalities, users often have no choice but

[1]Parts of this work were carried out while Katarzyna Olejnik and Mohammad Emtiyaz Khan were working at EPFL and Kévin Huguenin was working at LAAS-CNRS.

to overshare personal information. A better trade-off can be achieved by providing users with additional decision types, where sensitive information is only partially revealed to apps in exchange for some utility, i.e., *data obfuscation*. For instance, to check the weather forecast, a user could reveal her approximated location instead of the precise one. In our user survey (Section V), 73% of the participants reported finding data obfuscation useful.

To address the aforementioned limitations, we propose "Smart Permissions" ("*SmarPer*"), an advanced permission mechanism for Android with three main goals: context-aware permissions, automatic decision-making at runtime, and data obfuscation. *SmarPer* follows a platform-agnostic design where apps' sensitive requests are intercepted at runtime and users are prompted for a decision, i.e., allow, obfuscate, deny. By observing users' responses, *SmarPer* progressively learns to predict and make decisions on behalf of users. It should be noted that the goal of *SmarPer* is to *mimic* users' decisions, not to make privacy-preserving decisions or to find a balance between utility and privacy. In other words, if a user makes poor privacy decisions, *SmarPer* will do the same.

SmarPer relies on machine learning to predict users' permission decisions. Instead of using a multi-class classifier approach, as prior work in this area[2] [8], [9], we model the problem as a *linear regression* problem, using a one-dimensional privacy-preference function that outputs the degree of privacy of each user for each request (allow<obfuscate<deny). Specifically, by using a set of contextual features, we use a *Bayesian linear regression model (BLR)* to fit a linear regression to each users' decision data. This model has several advantages: It is lightweight enough for smartphones, and it is well suited for limited amounts of training data. Also, by training directly in the smartphone a model per-user, it preserves users' privacy.

We use an implementation of *SmarPer*, based on XPrivacy [4], to collect at runtime permission decisions from 41 participants[3]. Each participant used *SmarPer* (in fully manual mode) for a period of at least 10 days. Unlike previous studies [2], [10], ours relies on decision data collected in real conditions, i.e., participants using *SmarPer* daily on their own or loaned devices with their own apps. In total, we collected 8,521 unique permission decisions, along with 32 raw contextual features per decision (e.g., time, location, app name, etc.). Using this unique data set, our model achieves a mean absolute error (MAE) of 0.22 ($\pm$0.03)[4] and a mean incorrect classification rate (ICR) of 0.20 ($\pm$0.03), i.e., a mean correct classification rate (CCR) of 80% ($\pm$3%). This represents a mean relative improvement of 55% for the MAE and 50% for the ICR over a static policy baseline, where participants manually define permission decisions, i.e., the model used by current permission systems. Our results show that it is possible to learn the decision patterns of users with good accuracy, even when training data is scarce, and that contextual information is key for such a task.

We also implemented in *SmarPer* obfuscation techniques for four data types: location, contacts, storage, and camera. During our data-collection campaign, we evaluated three of these techniques with our participants. Our results demonstrate the importance of obfuscation: *Obfuscate* accounted for 27% of the total number of decisions collected and, in our user survey, 80% of the participants stated that they would like to obfuscate additional data types. Few users reported compatibility problems with apps. We believe this is the first evaluation of obfuscation techniques in smartphones on this scale.

It is important to mention that there are two key parts in the *SmarPer* project. First, modeling users' permission-decision patterns by using contextual information and data obfuscation. This part requires a user study to collect the decision data required to assess the *potential* of our machine learning approach. Second, evaluating our machine learning approach in practice (including user perception and the use of *SmarPer*'s features), through a new field experiment and user study. In this paper, we present the results associated with the first part. We are currently working towards the second part.

Our main contributions are as follows:

- **Design and partial implementation of *SmarPer***. We present a platform-agnostic design to support context-aware and automatic decisions at runtime, and data obfuscation. Our implementation, publicly available as an open-source project [11], offers runtime collection of permission decisions and associated contextual features, and data obfuscation for four data types.
- **Unique data set of permission decisions**. We collected 8,521 runtime permission decisions and their context from 41 participants. We believe this is the largest and most realistic data set of this type. After the approval of our university's ethical committee, we made a sanitized subset of this data set publicly available [11].
- **Evaluation of the potential of automatic prediction techniques for permission decisions**. We use an adapted linear regression model and demonstrate that it achieves significant performance improvements over two carefully chosen baselines. Our results show that contextual information is key to accurately predict permission decisions. In addition, we show that a per-user model has better performance than a one-size-fits-all model, as the former is able to better capture users' unique privacy preferences.
- **Machine learning framework.** We provide a framework for carefully training and comparing different context-aware models that predict permission decisions. The framework's source code is also available online [11].
- **Implementation and evaluation of data obfuscation**. We develop obfuscation techniques for four data types in *SmarPer* and evaluate them in our data-collection campaign. This is one of the first and largest evaluations of obfuscation in smartphones with real users.

The remainder of this paper proceeds as follows. Section II highlights related work in the area of mobile permission

[2]Note that prior work uses machine learning to predict users' static permission configurations instead of runtime permission decisions.

[3]This user study was approved by our institution's IRB (ethical committee).

[4]On a scale from -1 to +1; thus, the maximum value for the MAE is 2.

systems. Section III presents *SmarPer*'s design goals and architecture. Section IV describes our *SmarPer* implementation. Section V explains our data-collection methodology and the resulting data set. Section VI describes our machine learning methodology to predict users' decisions using contextual information and presents the results of our performance evaluation. Section VII further discusses our data set and machine learning results, and the limitations of our study. Finally, Section VIII presents our concluding remarks and future research directions.

II. Related Work

Mobile permission systems have several limitations that cause users [12], [13] and developers [14] difficulty in understanding and managing them. Researchers propose different extensions to current permission models to provide users with more control and better management [8], [15]–[19]. Yet, most of these approaches do not support contextual information in their access control policies (i.e., static policies). The lack of context-awareness in mobile permission systems has also been addressed in previous work [1], [2], [20]–[23]. Still, most of the proposed solutions are not practical for average users, as they require manually defining context-aware policies for each (app, permission, inferred context) tuple. In contrast, our work focuses on the automatic inference of these policies from users' decision-making behavior in different contexts. Another limitation is the lack of decision-granularity i.e., only "allow" or "deny" decisions. To address this issue, researchers propose sending fake or obfuscated data to apps [18], [22], [24], [25]. Yet, most of these solutions have not been evaluated with users in real scenarios. To fill this gap, we implement obfuscation techniques for different data types in Android and perform one of the first evaluations of obfuscation with real users.

With the increasing number of apps and data types, another important area of study is helping users to efficiently manage permissions. Machine learning has been used to automate decision-making in other areas such as location-based services and social networks [26]–[29]. In the area of mobile devices, researchers propose crowdsourcing [19] and machine learning to help users manage permissions [8], [9], [30]. For instance, Lin et al. and Liu et al. identify a small number of "privacy profiles", using clustering techniques that could be used to facilitate static permission configuration for different types of users [8], [9], [30]. Also, Liu et al. [8] show that a binary classifier can predict users' static permission decisions with high accuracy. These works, however, focus on static permission policies that do not change over time, i.e., no context-awareness, and they rely on a one-size-fits-all approach, i.e., training a single model with data from all users.

Closer to our work, Wijesekera et al. [2] propose the use of permission decisions at runtime to provide users with contextual information to make more informed decisions. To reduce a user's load, they conclude that a mechanism should infer when to prompt users or automatically block app requests (note that this is not exactly the same as predicting users' decisions). The authors show how a one-size-fits-all mixed-effects logistic regression model can be used for this purpose with good accuracy, using a small data set of users' decisions collected in semi-realistic conditions, i.e., 673 decisions from 36 users collected offline during an exit survey. We extend this line of work in several ways. First, we demonstrate that it is possible to predict users' permission decisions with great accuracy, even when we consider an additional decision type (i.e., obfuscate), and that contextual information is key for doing so. Second, we show that a per-user model is significantly more accurate than a one-size-fits-all model, due to users' unique privacy preferences. Third, we provide an experimental framework and methodology for carefully comparing the performance of different machine learning algorithms that predict decisions using contextual information. Fourth, we use a unique and substantially larger data set of permission decisions *per user*, collected in real conditions (i.e., 8,521 decisions from 41 users collected at runtime in users' smartphones), and we describe the many challenges faced when doing so. Fifth, we provide a design and partial prototype of a mechanism for predicting and automating permission decisions and propose an approach for automating permission decisions. Sixth, we design and implement obfuscation mechanisms in our prototype and evaluated them with real users.

III. SmarPer

We address two important limitations of the current permission systems: the use of static policies that do not capture users' privacy needs in different contexts, and the sub-optimal trade-off between privacy and utility. We propose *SmarPer*, an extension to Android's permission system that supports dynamic and automatic decision policies inferred from users' behavior, and that provides finer-grained decisions, i.e., allow, obfuscate, and deny. *SmarPer* provides a feedback loop where users are initially prompted for permission decisions, and over time *SmarPer* learns users' decisions patterns and makes decisions on behalf of the users. *SmarPer* can even adapt to changes in users' privacy posture. Note that, though *SmarPer* targets Android, its concepts and design are platform-agnostic.

A. Threat Model

We focus on the case of privacy-invasive apps, that access private data (e.g., location, camera) about users through the dedicated APIs calls of the mobile OS. We do not address the cases where the threat comes from the OS itself or from apps that use native code or security breaches to access private data. We assume that the considered apps access, in some cases, more information than they actually need to provide the features (and the associated quality of service) the users actually need; this constitutes a privacy threat for the users.

B. Design Goals

SmarPer's design follows three main goals: *Context-aware permissions*, to support dynamic permission policies that change according to users' context; *Automatic permission decisions*, to predict and make permission decisions at runtime on behalf of the user and reduce users' load; and *Data*

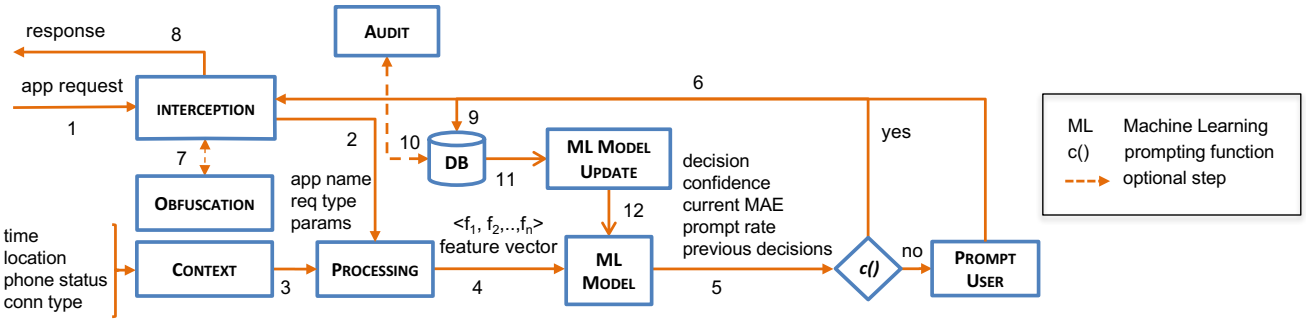


Fig. 1. *SmarPer* architecture. Intercepted apps' requests together with contextual information are used as input to a machine learning model that predicts a decision. After step 5, a function $c()$ takes as input the decision and other parameters to decide if the decision should be made automatically or if the user should be prompted. All decisions and contextual information are stored for continually training the model.

obfuscation, to reduce the sensitive information revealed to apps while maintaining some level of utility.

We also consider the following deployability goals: *Efficiency*, to support *SmarPer* in hardware-constrained mobile devices; *Privacy*, to guarantee that users' decisions and contextual information gathered will not compromise users' privacy; and *Flexibility*, to enable users to configure different obfuscation techniques, privacy preferences, and learning rates, and to correct possible prediction errors.

C. System Flow

Figure 1 shows the general architecture and data flow of *SmarPer*. As stated before, *SmarPer* follows a feedback loop approach. First, *SmarPer* intercepts a privacy-sensitive app request (1), e.g., Android API call for location. The app name, request type, and parameters in the API call are sent for processing (2). Information about the current context is collected from the device (3). This includes information about the app (e.g., whether it is in the foreground or background), smartphone state (e.g., whether the screen is locked), and smartphone's sensor information (e.g., location provided by the GPS). For more details, see Section IV-C. All of this information is processed into a feature vector that is input to the machine learning model for prediction (4). Given the user's past decision and contextual data, the model predicts whether the user would allow, obfuscate, or deny the request; and it outputs the predicted decision, along with other parameters such as the estimated confidence (e.g., mean and variance) (5). A function $c()$ take as input the predicted decision and estimated confidence, as well as other parameters such as current MAE, prompt rate, and previous decisions, and it determines if the predicted decision should be used (i.e., automatic decision) or if the user should be prompted (Figure 2). The function $c()$ can be adjusted by the user to regulate the number of prompts.

When *SmarPer* has a decision (manual or automatic), it prepares to respond to the requesting app (6). If the decision calls for obfuscation, obfuscation is performed on the requested data before returning it to the app (7), e.g., reducing the precision of the location. *SmarPer* then responds to the app with the requested data (8). Finally, the decision, contextual information, and whether the decision was manual or automatic are recorded in the *SmarPer* database (9). Optionally, the user can review recent automatic decisions and correct them if they are wrong (10). This user feedback is also incorporated in the model (11 and 12). More specifically, the corresponding corrected decisions are added to the training set and the model is updated, possibly with higher weights. The user can set a parameter to determine the cost-sensitivity, i.e., the user can express which type of error they are more willing to tolerate (oversharing or undersharing).

IV. SmarPer Implementation

We implemented a partial version of *SmarPer*, compatible with Android 4.0.3 to 5.1.1, to evaluate obfuscation techniques and to collect permission decisions and their associated contextual information. During our field experiment (Section V), our *SmarPer* implementation operated in full manual mode: There were no automatic decisions or learning from users' behavior, users were always prompted at runtime for decisions. The performance of our machine learning model was evaluated offline using the data collected (Section VI). Such an offline machine learning approach, used by other works in this area [2], [8], [23], enables the use of a larger variety of machine learning tools and analysis techniques to assess the potential of *SmarPer*. We focused on a robust implementation to avoid interfering with the OS and apps and it is available as an open-source project under a GPLv3 license [11].

A. Request Interception

Android and other popular mobile platforms do not provide a native API to mediate apps' requests, as suggested by Heuser et al. [17]. Therefore, we took a rooted-device approach to dynamically intercept apps' requests, without modifying the OS; rooting a device is easier than flashing a new firmware, and there are millions of users with rooted devices [8]. To intercept privacy-sensitive API calls, our implementation builds on the open-source permission tool XPrivacy v.3.6 [4].

XPrivacy is a module of the Xposed framework [31], a general framework that lets users install modules to modify the look and feel of their smartphone. It requires root privileges

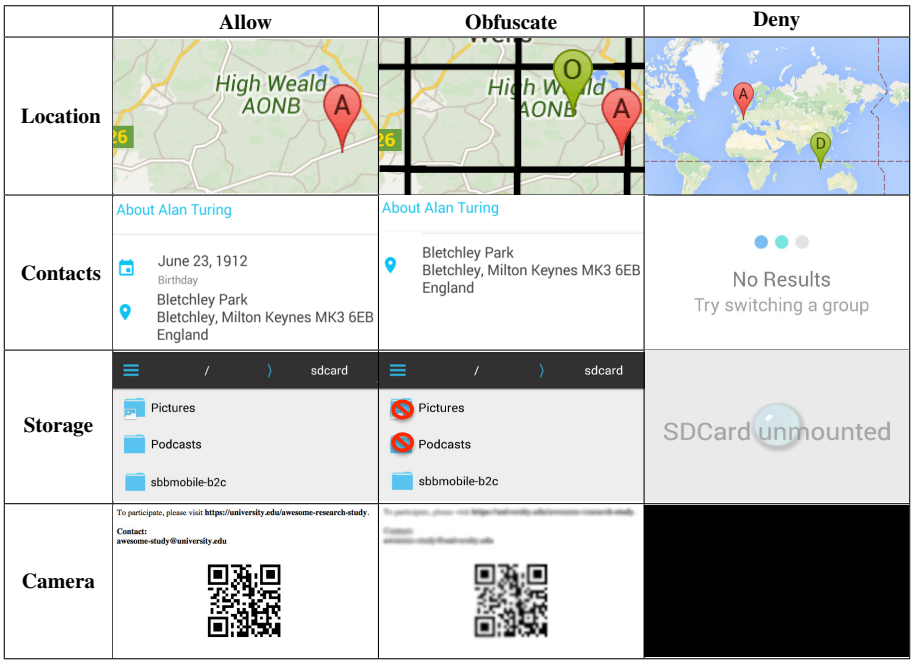


TABLE I

EFFECT OF EACH DECISION TYPE (ALLOW, OBFUSCATE, DENY) ON DIFFERENT DATA TYPES IN *SmarPer*. WE IMPLEMENTED OBFUSCATION FOR LOCATION, CONTACTS, STORAGE, AND CAMERA AND EVALUATED THEM IN A FIELD EXPERIMENT WITH REAL USERS.

in order to change the *app_process* executable to add an additional library (Xposed). This enables developers to hook Android API calls and execute code before or after API call execution, to modify OS or apps' functionalities. All of these modifications are done in memory [32]. Note that there is a chance that this request-interception approach could be bypassed by (malicious) apps to avoid *SmarPer*; however, such threats are out of the scope of this work. The main methods we propose in this work, essentially obfuscating sensitive data and predicting users' decisions, are independent from the underlying request interception technique used.

B. Data Obfuscation

In *SmarPer*, users can allow, obfuscate, or deny access to their private data. By using the Xposed framework, we were able to modify the parameters and return values of sensitive API calls before or after it executes. The "allow" case is straightforward: We allow the API call to execute without modification. For "obfuscate", we remove some level of detail from the returned data. For "deny", we return fake values.

We faced multiple challenges while implementing obfuscation techniques. First, obfuscation is data dependent. Hence, different techniques must be used for different data types; and data types can have more than one obfuscation technique. For instance, for the camera, we can reduce image resolution or blur people faces. Hence, we envision a community of privacy-conscious developers implementing obfuscation "plugins" for *SmarPer* in the future. Second, the utility of each obfuscation technique depends on the type of app and use case. For example, to reduce the risk of unauthorized pictures, a QR code scanner app can still work if obfuscation only blurs people faces from images. Note that the privacy implications of data obfuscation depend on the type of data, the data itself, and on the background knowledge available to the adversary (e.g., a service provider or an ad network). Third, obfuscated data could cause the app to crash or behave unexpectedly [24]: e.g., we noticed that the WhatsApp messenger app will not display correctly the name of the user's contacts if we obfuscate access to the contacts database. Fourth, there is no native support for obfuscation in mobile platforms. Therefore, to implement obfuscation techniques, we need to understand the low-level details of how the OS processes each data type. We implemented obfuscation techniques for the following four data types (Table I):

Location: We implemented location obfuscation by discretizing the Earth into 10km by 10km areas. Instead of returning the user's exact location, *SmarPer* returns the coordinates of the center of the current area the user is in, as shown by the green icon in the first row of Table I. The size of these areas is easily configurable. A more privacy-consistent solution would use a variable size based on the population or point-of-interest density. More advanced location privacy-protection mechanisms (LPPM) could also be considered such as [33] (optimal grid-based obfuscation) or [34] (differential privacy/geo-indistinguishability), as static obfuscation, based on a fixed grid, is known to be vulnerable to attacks [35].

Apps generally ask for the user's location to tailor some features to the user's location, e.g., ads or the current weather. With this approach, apps can provide the same level of utility while the user's privacy increases, as their exact location is not revealed. For the deny decision, a fixed set of coordinates are returned. The level of privacy-protection provided by such

data obfuscation techniques can be captured by the (difference of) accuracy of known inference mechanisms on location data (e.g., filling the gaps in an obfuscated location trace [36] and inferring activity preferences [37] and interests [38]).

Contacts: Implementing obfuscation for contacts was particularly difficult. Android stores contact information in a SQLite database. Apps can query this database to request any information they need, as long as they have the READ_CONTACTS permission. With *SmarPer*, we have access to the actual queries that apps make to the contacts database, as well as a Cursor object with the returned data that we can modify before it is returned to the app. To implement obfuscation, we filtered out rows from this Cursor that are not *names, phone numbers, postal addresses*, or *e-mail addresses*. Yet, because apps have great flexibility to query this database, a column identifying the type of the returned information is not always present in the Cursor. This means that we might not know what type of private data we are looking to filter out in the result. If we have the type information, we filter out rows that are not names, phone numbers, postal addresses, or e-mail. Otherwise, we check all the columns containing data in the returned Cursor with regular expressions for these four allowed types. For phone number and e-mail we use the standard Android API calls *PhoneNumberUtils.isReallyDialable()* and *Patterns.EMAIL_ADDRESS*. Rows that do not match the regular expression for name, phone number, postal address, or e-mail are discarded from the result before it is returned to the app. For the deny decision, *SmarPer* simulates an empty address book by returning an empty Cursor.

Most apps request access to contacts to find user's friends already registered in the service. For this purpose, name, phone number, postal address, or e-mail address should suffice. Users can enter a variety of other (potentially sensitive) information about their contacts into the contacts database, such as birthdays, relationship to the user, and employer. Revealing this information to an app does not have a clear use case. By revealing only names, phone numbers, postal addresses, and e-mail addresses, we maintain utility for the majority of apps and reduce the amount of sensitive information revealed.

Storage: To implement obfuscation for storage, we restricted access to the Android Public directories (accessible to all apps with the READ_EXTERNAL_STORAGE permission) – Pictures, Music, Movies, and DCIM (Camera pictures) that, ironically, actually contain private data. For this purpose, *SmarPer* returns a "FileNotFoundException"; this is an exception that an app should be prepared to handle: It can happen that these files really do not exist. We also filter out these URIs from queries made to MediaScanner, a service that keeps track of all the user's files on the device. For the deny case, *SmarPer* simulates that the external storage is unmounted.

To create a cache on the external storage, some apps request the WRITE_EXTERNAL_STORAGE permission (which implicitly includes READ_EXTERNAL _STORAGE permission). This functionality is preserved with our obfuscation technique. However, a curious app which wants to sift through the user's photos will not be able to do so. Some apps, however, need write access to the Public directories, to save new data there, e.g., photos. If this is the case, the user will need to allow access to storage to preserve functionality. Evaluating the privacy protection of such an obfuscation technique, or more generally the privacy risks of accessing data on users' SD card, highly depends on the data.

Camera: We obfuscated two aspects of the camera: the camera preview (i.e., when the user opens the camera but has not taken a photo yet) and taken photos. For both of these, we reduce the resolution of the returned image by scaling down and then scaling back up to the original size. The scaling factor is configurable. For the deny case, we return a black image.

Apps with access to the camera pose a considerable threat to user's privacy: They can take photos without notification, as long as the camera preview is open. Thus, we distort the returned images. Yet, it is still possible to scan QR codes. Therefore, QR code scanner apps maintain their utility and the user maintains privacy. A more advanced alternative to blurring the whole image would be to only blur or remove detected faces or detected text [39] and/or window blind the image [40]. A potential way to evaluate the privacy protection of such obfuscation techniques is to evaluate the performance of a standard library in inferring private attributes (e.g., gender [41], age,[5] ethnicity [42]) of the user, as well as the context (e.g., emotions,[5] activity) from the captured video/photo.

Future data types: In general, obfuscation techniques can be implemented for most data types. As part of our future work, we plan to implement obfuscation for other data types such as microphone data (e.g., filtering out frequencies corresponding to the human voice) and calendar data (e.g., filtering out information from events such as location and invited guests).

Note that the implemented obfuscation techniques are somewhat simple and not optimal: They may offer only limited protection for users' privacy (depending on the data and on the background knowledge available) and/or limit the utility of the app. Yet, they offer more control to the user. They also are a good starting point for evaluating users' perception of obfuscation in realistic and easy-to-understand scenarios. We believe that obfuscation methods should be designed by taking into account the specifics of the considered data, how it is used by different (categories of) mobile apps, and the privacy implications of the data disclosure, based on research results in order to determine to which extent users can still be tracked, identified, or profiled from the disclosed (obfuscated) data. Designing such techniques is a research problem on its own (for each type); we leave it to future work together with the evaluation of the privacy implications of data obfuscation.

C. Contextual Information

For each permission decision, using standard Android API calls, *SmarPer* collects the associated contextual information. Using this information, we selected raw contextual features that we estimated have an impact on users' permission decisions. This list is not exhaustive: There could be other

[5]Microsoft's Face and Emotion APIs, https://www.microsoft.com/cognitive-services/. Last visited: Feb. 2017.

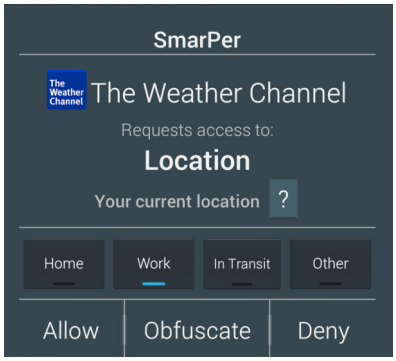


Fig. 2. *SmarPer* permission prompt. The Weather Channel app requests access to the user's location. Clicking on the question mark shows information about the effect of the different decision types. Below that, we can see the semantic location and decision buttons.

important contextual features. In Section VI, we show a subset of the most relevant features across all participants.

In total, we selected 32 raw features for our machine learning analysis (Section VI):

- *App information (6):* UID, GID, package name, name, version, and Google Play Store category.
- *Foreground app information (3):* package name, name, and activity.
- *Request information (4):* XPrivacy category, method name, parameters, whether it is dangerous (i.e., denying it may break the app).
- *Decision information (4):* type, current time, time to make the decision, and whether the decision has been modified by the user.
- *Device status (14):* screen in interactive mode, screen locked, ringer state, headphones plugged, headphone type, headphones with a mic, battery percent, charging state, charger type, network connection type, dock state, latitude, longitude, and location provider.
- *Semantic location (1)*: users are asked to choose a label for their current geographical location. For usability purposes, only four labels are used (see Figure 2).

D. Data Collection Considerations

The purpose of our current *SmarPer* implementation is to collect at runtime users' permission decisions (Section V). We want to collect as many users' decisions as possible but not to overwhelm users or cause habituation to the prompts (Figure 2). Otherwise, we could end up with noisy and unreliable data. To address these issues, we added the following mechanisms to *SmarPer*:

Prompt rate-limiting: As previous work [2] and our evaluation shows, most apps make a large number of requests for users' data. Hence, it is not practical to prompt users each time an app makes a request. To address this problem, we implemented the following rate-limiting policy for the apps and data types targeted in our study (Section V). If the user is using the app (i.e., foreground app), *SmarPer* does not limit the number of prompts associated with this app. If the user is not using with the app (i.e., background app), *SmarPer* only permits one prompt every 10 to 20 minutes, sampled uniformly from that interval. If the rate limit has been exceeded, *SmarPer* takes the most recent decision for the same type of request. If no previous decision exists, *SmarPer* prompts the user. Also, *SmarPer* caches each decision for one hour to avoid prompting repeatedly for the same type of request. If the user is not using the smartphone, *SmarPer* applies the previous decision; otherwise, *SmarPer* allows the request. Requests associated with apps and data types not in our list are always accepted.

Non-interruption policy: For some activities (e.g., typing, calling, taking a photo), it is better to not interrupt users with prompts, as it can be problematic and lead to noisy data. Hence, *SmarPer* does not interrupt the user in such situations; instead, it uses the previous decision for the same type of request. If there is no previous decision, the request is allowed. *SmarPer* checks if the user is calling using the *TelephonyManager* API or if the user is typing or taking a picture by intercepting API calls such as *InputMethodManager.showSoftInput(), Camera.open(), and Camera.release()*.

V. Collecting Permission Decisions

To support automatic decisions in *SmarPer*, we need data on how users make permission decisions at runtime and the contextual information associated with such decisions. This data is used to train a machine learning model that captures the permission-decision patterns of each user (Section VI). Unfortunately, data sets from previous works do not satisfy our requirements. For example, they do not include runtime permission decisions [8], [30]. Other data sets include runtime decisions but were collected in non-realistic scenarios [2], [10]. Initially, we considered using the data set from Wijesekera et al. [2], as it seemed to match our requirements. But, we determined it was not appropriate for our goals, as participants' decisions were collected offline during an exit interview (i.e., the context at request time is different from the one at decision time), and the number of decisions per participant is limited (i.e., 10-15 decisions per participant) for training a machine learning model per participant. Hence, we decided to conduct a data-collection campaign using our partial *SmarPer* implementation and build our own data set.

Besides the technical requirements, a key challenge for our data-collection was to gather, in a limited period of time, enough data for our machine learning analysis without overwhelming users with prompts or causing prompt habituation. Moreover, the data collected can be very sparse [8], given the great variety of apps, permissions, and contextual information available. Hence, besides the mechanisms described in Section IV, we also decided to collect decisions from only a subset of apps and data types. We chose a set of popular apps from the US Google Play Store that belong to different categories and make requests for at least one of the following data types: *location, contacts, and storage*. By using popular apps, we increased the chance of (1) collecting more decision-data from each app during the study (i.e., popular apps are used more often), and (2) having more than one participant

using each app (to facilitate comparisons). This resulted in a total of 29 apps: Accuweather, Amazon, Candy Crush Soda Saga, Clash of Clans, Dropbox, Evernote, Facebook, Fitbit, iHeartRadio, Instagram, Kik, Lyft, Runtastic, Shazam, Skype, Snapchat, Soundcloud, Star Wars: Galaxy of Heroes, Subway Surfers, The Weather Channel, TripAdvisor, Twitter, Uber, Viber, Walmart, Waze, WhatsApp, Wish, and Yelp.

A. Methodology

Here, we describe the steps in our data-collection campaign. **It is important to note that the data collected contains no personally identifiable information of the participants and that our study was approved by our institution's IRB (i.e., ethical committee).** In addition, all the data collected is securely stored and can only be accessed by authorized researchers from our institution.

1) Recruitment: We recruited remote and local participants through posts on online forums and flyers on our campus. Participants were required to be at least 18 years old, be regular Android users, be regular users of at least two of the apps selected for our study, and have reliable cellular and WiFi Internet connectivity. We offered a $50 gift card as a reward.

2) Setup, Training, and Entry Survey: Both local and remote participants had access to *SmarPer*'s training material (e.g., written instructions and video tutorials) hosted in our server. Before starting the study, participants had to agree to our consent form and complete an entry survey. In this survey, we asked participants some demographic questions and some questions to estimate their general level of privacy concerns. We made use of the IUIPC scale [43], as well as some questions of our own design, adapted to the smartphone environment. Remote participants used their personal, *SmarPer*-compatible smartphones (i.e., rooted Android 4.0.3-5.x devices). Local participants had the option of using their personal devices or using a smartphone provided by us, notably Motorola Moto G 2^{nd} or 3^{rd} generation devices with Android 4.4.5. In the latter case, to guarantee normal use patterns, local participants met with one of our researchers to set up the loaned device: transfer the participant's SIM card and data, and install the participant's apps. We also asked participants to treat the loaned smartphones as their personal devices. This step is thus similar to the one followed by Wijesekera et al. [2]. In total, we loaned smartphones to 29 participants. We also explained to participants the functioning of *SmarPer*, in particular, the effect that the three decision types have on the targeted data types and their purpose, i.e., data minimization.

It is possible that our *SmarPer* training influenced participants towards a more privacy-preserving behavior. Such bias is difficult to avoid when evaluating a privacy mechanism with real users. We cannot properly evaluate *SmarPer* without first explaining concepts such as permission prompts and data obfuscation. Nevertheless, such bias is unlikely to affect our analysis, as our goal is not to estimate if *SmarPer* makes participants more privacy-conscious. Instead, our goal is to model participants' unique privacy preferences when prompted for permissions in different contexts (even if there is a bias) and their attitudes towards obfuscation. Our scenario is similar to the one of runtime permission-prompts in Android 6+ and iOS, where users are explained first how permission prompts work and their purpose.

3) Data-collection: Participants agreed to run *SmarPer* on their personal or loaned smartphones for at least 10 days. During that period, *SmarPer* prompted participants for permission decisions (Figure 2) associated with our selected apps and data types. The goal was to collect at least 75 decisions per participant and the contextual information associated with each decision. If this targeted number of decisions was not reached after 10 days, participants were encouraged to continue the study for some additional time until it was reached (to avoid bias, we did not explicitly ask participants for more decisions). Every day, the decision data was automatically uploaded to our server over an encrypted connection. Hence, we were also able to monitor for problems with *SmarPer* or if users were not actively using the smartphone. In the latter case, we contacted the participants to remind them about the rules.

4) Static Policies and Exit Surveys: At the end of the data-collection, all participants were required to complete two additional surveys. In the *static policy survey*, for each app monitored during the study, we asked the participant to define what *static decision* (i.e., allow, obfuscate, deny) they would grant to access each of the monitored data types (i.e., location, contacts, storage). The purpose of this survey is to capture how participants would configure permissions on their personal smartphones by using the interface provided by current permission systems (e.g., Android 6+). The data from this survey was used as a baseline in our analysis (Section VI).

In the exit survey, we asked participants about their experience with *SmarPer* and their aptitudes towards using automatic decisions, data obfuscation, and contextual information in mobile permissions. We conducted supplementary interviews with selected participants to better understand the reasons for their decisions during the study. After completing both surveys and passing the data consistency check (described next), participants were rewarded with a gift card.

5) Data Quality: We performed different checks to validate the quality of the data submitted by participants. First, we checked that participants did not respond to prompts too rapidly, i.e., at least two seconds elapsed before they chose their response. The fact that participants had to touch the screen twice per prompt (Figure 2) reduced the chances of quick random decisions. Second, we checked the consistency of the semantic locations reported by the participants, by comparing the semantic labels with the actual coordinates recorded at decision time. For example, if a participant reported "home" in two or more geographical locations, it is likely that the participant provided false information. No users violated the above conditions significantly enough to warrant being removed from the study. Third, for each participant, we removed the first and last five decisions from the data set, as they were made during the familiarization with *SmarPer* and when participants returned the loaned devices, i.e., noisy data.

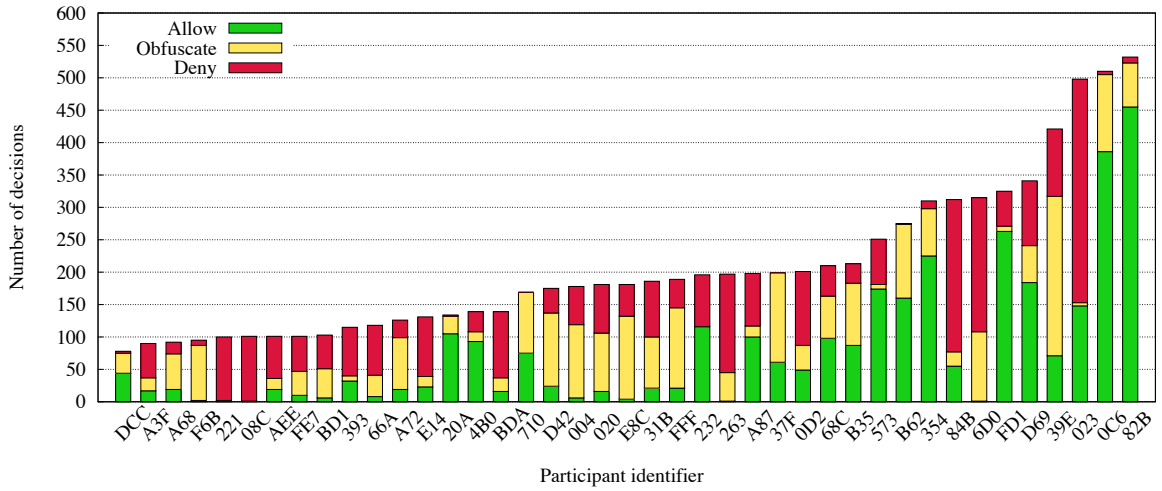


Fig. 3. Total number of decisions per participant, including the distribution of the decision types. The difference in the number of decisions is mostly due to the participant's behavior, number and type of apps used, and days in the study.

6) Data Preprocessing: We converted some of the raw contextual features (Section IV-C) before our machine learning analysis (Section VI). First, categorical features (e.g., app name and category) were converted into dummy features [44], because techniques, such as linear regression, do not work directly with categorical features. These dummy variables take the value 0 or 1 to indicate the absence or presence of some categorical effect. Second, we computed five additional features based on the raw features collected: whether an app is in the foreground, day of week, part of day (i.e., morning, afternoon, evening, night), battery charge percentage, and day/month/year. We ended up with a total of 37 features.

B. Data Set Details

A total of 47 participants joined our data-collection campaign; 41 completed it successfully. Overall, we collected around 4.82 million apps' requests for private information. Of these, we prompted participants for 8,521 manual permission decisions. The rest corresponds to requests associated with apps and data types outside the scope of our study.

1) Demographics: From the 41 participants that completed the study: 17 (41%) were female; 29 (71%) were in the 18–25 range and 12 (29%) were in the 26–50 range; 12 (29.3%) were undergraduate students, 23 (56.1%) were graduate students, 3 (7.3%) worked in scientific services, 1 (2.4%) worked in education, 1 was unemployed, and 1 did not disclose their occupation. Participants reported being active smartphones users (1-3 hours/day) and long-term Android users (2-5 years).

In the entry survey, participants scored high on a 5-point IUIPC scale for control, awareness, and collection of private information. These results indicate that most of our participants have a high level of privacy concern. Participants reported high concern regarding apps accessing their contacts, camera, or storage. Surprisingly, participants were less concerned about apps accessing their location.

2) Exploratory Analysis: In our final data set, allow, obfuscate, and deny account for 42%, 27%, and 31% of the total number of decisions, respectively, thus showing a balanced distribution of decision types. Figure 3 shows the total number of permission decisions per user, including the distribution of decision types. Participants chose for contacts: 65% allow, 24% obfuscate, and 11% deny; for location: 25% allow, 27% obfuscate, and 48% deny, and for storage: 35% allow, 31% obfuscate and 34% deny. We conclude that participants are more likely to allow contacts and deny location requests. These results contradict the concern levels reported by participants in the entry survey (Section V-B1), where they stated to be more concerned about apps accessing their contacts than their location (i.e., "privacy paradox" [45]).

In Figure 3 we also notice that some participants were significantly more active than others. This difference is due mainly to the participant's individual behavior, the number and type of apps used, and the number of days in the study (66% of the participants completed the data collection in less than 15 days). Participants were prompted a median of 17.3 times per day and each prompt was completed in a few seconds. These numbers show the effectiveness of our rate-limiting mechanisms (Section IV-D). For comparison, participants in [10] were prompted at least 10 times a day and each prompt required 2-5 minutes to complete. We observe that the distribution of decision types varies considerably across participants, indicating the unique privacy preferences of each user and hinting at the difficulty of predicting permission decisions. Figure 4 shows this difference more clearly by depicting the initial, middle, and last 12 decisions of a subset of participants. We observe that the distributions are reasonably stable over time per participant, especially after some initial period where some participants changed their preferences. Participants are vertically grouped according to their privacy profile: utility-concerned (top), somewhat-privacy-concerned (middle), and privacy-concerned (bottom).

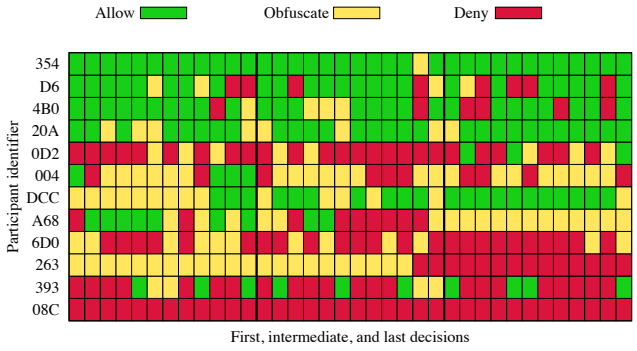


Fig. 4. Decision patterns over time for a subset of participants and decisions. Each row shows the first, middle, and last 12 decisions of each participant. We can observe the different privacy behavior of participants. Participants are grouped vertically in three categories: utility-concerned (top), somewhat-privacy-concerned (middle), and privacy-concerned (bottom) participants.

On average, participants used $4.2(\pm 2.0)$ apps from our list of apps. In our data set we have data from 23 apps out of 29 in our list. Note that the fact that participants used a small number of apps is beneficial for our analysis, as it enabled us to collect more decision data per used app during the study, hence reducing data sparsity. Figure 5 shows apps with more than 10 decisions and more than 1 participant. We can see that the three most popular apps are WhatsApp, Facebook, and Skype with 36, 33, and 19 participants, respectively. The difference in the number of decisions is due not only to the number of participants per app, but also to the type of app and how active each participant was. More details about the number of decisions per app are shown in Table II (Appendix).

VI. Data Analysis: Predicting Decisions

In this section, we present the machine learning analysis of our data set. We describe and compare various methods for context-aware and automatic permissions. Our goal is to predict users' preferred privacy levels for a new permission prompt, given their past decisions and associated context.

A. Problem Statement

We index users by u and permission requests by i and j. We denote user u's decision for the i'th permission request by $y_{ui} \in \{$"Allow", "Obfuscate", "Deny"$\}$. We denote the context of the permission request by a feature-vector $\mathbf{x}_{ui} \in \mathcal{X} \subset \mathbb{R}^D$ and the time the request was made by t_{ui}. We denote the user's past data before time t by $\mathcal{D}_{u,t}$ to be the set of all decision pairs $\{y_{ui}, \mathbf{x}_{ui}\}$ made at time $t_i < t_u$. Our goal is the following: given a user's past decisions $\mathcal{D}_{u,t}$, predict the users' decision y_* at a future time given a feature vector $\mathbf{x}_*$.

We focus on two important aspects: (1) we can *learn* to predict permission decisions, (2) context helps us to do so. We also show that, as the amount of data per user increases (i.e., a higher t_u), our predictions improve much faster when we take context into account.

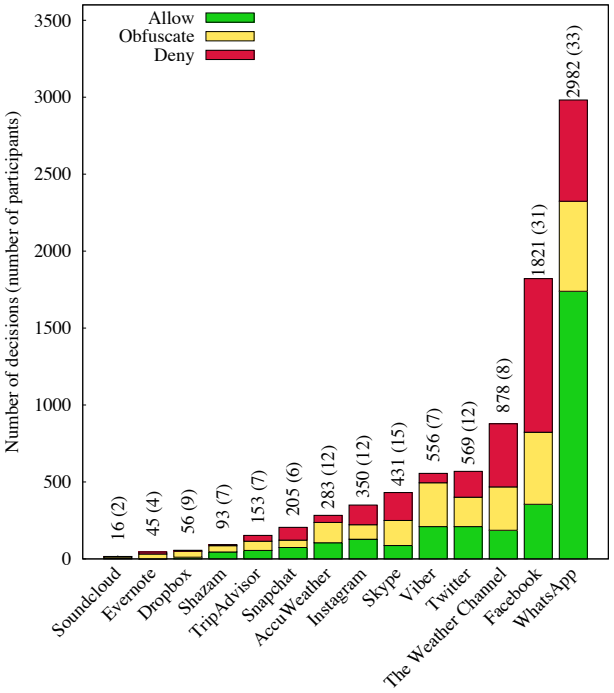


Fig. 5. Total number of decisions and participants (in parentheses) for apps with more than 10 decisions and more than 1 participant, including the distribution of the decision types. The difference in the number of decisions is due to the type and the popularity of the app.

B. Baselines

We use the following two baselines. The first baseline is referred to as the *static policy* method based on a survey completed by all participants (Section V-A4). Decisions collected in this survey are used as fixed predictions for permission requests. Over time, this method does not learn users' preferences and only takes part of the contextual information into account (i.e., apps' names and targeted data types). We expect this method to perform worse than a *dynamic* method that learns from users' behavior. This method approximates the current permission systems in Android 6+ and iOS.

Our second baseline ignores contextual information but learns the preference function from past data. We simply predict the most frequent decision made by the user until time t_u for all the new decisions. This method, although dynamic, might miss the contextual information associated with some decisions and might perform worse than a method that takes the context into account. We call this method *ZeroR*$_t$, because it is an extension of the ZeroR classifier [46].

C. Context-Aware Method

We compare our baselines to a method that learns from users' behavior and uses contextual information to predict. We model privacy preferences of the user u by a one-dimensional privacy-preference function $f_u : \mathcal{X} \to \mathbb{R}$. Given a feature vector $\mathbf{x} \in \mathcal{X}$, the value of function $f_u(\mathbf{x})$ indicates a degree of privacy: *a higher value indicates higher desire for privacy.* The

prediction is made by thresholding the preference function:

$$y_{ui} = \begin{cases} \text{“Deny”}, & \text{when} \quad \theta_1 < f_u(\mathbf{x}_i) \\ \text{“Obfuscate”}, & \quad\quad \theta_2 < f_u(\mathbf{x}_i) \leq \theta_1 \\ \text{“Allow”}, & \quad\quad f_u(\mathbf{x}_i) \leq \theta_2 \end{cases} \quad (1)$$

where θ_1 and θ_2 are two real-valued scalars. This is an example of the Random Utility Model and has been widely used to model users' preference functions (see [47]).

We use *Bayesian linear regression (BLR)* to model the preference function given the contextual information. The simplest model is to use a linear function:

$$f_u(\mathbf{x}) = \beta_u^0 + \boldsymbol{\beta}_u^T \mathbf{x} + \epsilon_{ui}, \quad (2)$$

where $\beta_u^0 \in \mathbb{R}$, $\boldsymbol{\beta}_u \in \mathbb{R}^D$, and ϵ_{ui} is the noise. We model both $\boldsymbol{\beta}_u$ and the noise ϵ_{ui} as i.i.d. Gaussian random variables.

Using the Bayes rule, we can compute the posterior distribution over predictions. However, the nonlinear function of (1) complicates this computation because it is not Gaussian. To simplify the computation, we make the following relaxation to (1): we fix thresholds[6] $\theta_1 = 0.5$ and $\theta_2 = -0.5$ and recode the decisions {“Allow”, “Obfuscate”, and “Deny”} as $\{-1, 0, +1\}$. This makes the decision y_{ui} Gaussian and then we can compute the posterior distribution in closed-form by using the Bayes rule. The BLR model outputs a real-value $\hat{y}$ which we threshold at θ_1 and θ_2 according to (1) to get the discrete-valued decision. The formulation presented in (1) and (2) enables us to use nonlinear models for f_u by using Gaussian Process models (GP). By simply changing the kernel matrix used, we can obtain a variety of nonlinear models (see Chapter 2 in [48]). This approach is similar to SVM algorithms, used in previous works [2], [8], with one important difference: the GP model gives us posterior probabilities for our predictions, unlike SVM where we need a two-stage procedure that requires large data to avoid overfitting (see Chapter 7 in [49]).

We note that our approach, BLR, differs from previous works that use only two classes “allow” and “deny” [2], [8]. For a two-class problem, the ordering does not matter, but for our problem it is clear that “obfuscate” requires less privacy than “deny” but more privacy than “allow”. Therefore, the choice of a one-dimensional function is justified, although this approach can be easily extended to a multi-dimensional function [50]. Another alternative would be to use multi-class classification (e.g., support-vector machines, classification trees), along with a cost-sensitive cost function [51]. Still, BLR is a reasonable first choice for small data sets, given its simplicity.

D. Error Measure

To reliably compare methods, we propose the performance error measure $\mathcal{E}$ to evaluate the performance of a method $\mathcal{M}$:

$$\mathcal{E}^t_{\mathcal{M}}(\mathcal{D}, \mathcal{D}_{test}) := \frac{1}{U} \sum_{u=1}^{U} \frac{1}{N_u} \sum_{i=1}^{N_u} \mathcal{L}(y_{ui}, \hat{y}_{ui|t}) \quad (3)$$

where $\mathcal{L}$ is a loss function, $\mathcal{D}_{test}$ is the set of test decisions y_{ui} for users $u = 1, \ldots, U$, $\mathcal{D}$ is the set of past decisions and contextual information $\mathcal{D}_{u,t}$ for these users, N_u is the number test decisions in $\mathcal{D}_{u,t}$, and $\hat{y}_{ui|t}$ are predictions computed by using $\mathcal{D}_{u,t}$ and the method $\mathcal{M}$. Note that the error measure is a random variable which depends on the choice of users in the test data and the data that contains the past decisions. This loss function is averaged over many users, therefore it penalizes methods that do not generalize well to many users at the same time. This is a better error measure than using one-leave-out methods that might show a high variance for different trials, as different users are selected in different runs.

We will use two types of loss functions. The first loss function is the popular 0-1 loss: $\mathcal{L}(y, y') = 1$ when $y \neq y'$ and 0 when $y = y'$. In this case, $\mathcal{E}$ is the standard incorrect classification rate metric (ICR). However, the 0-1 loss ignores the ordering between the three categories of the decision y: e.g., if we predict “allow” for “deny”, then it is more incorrect than predicting “obfuscate”, as the latter allows some degree of privacy. We use another loss function called mean-absolute error (MAE) that reflects these types of errors. We recode the decisions {“Allow”, “Obfuscate”, “Deny”} as $\{-1, 0, +1\}$ and we define the loss as follows: $\mathcal{L}(y, y') = |y - y'|$.

E. Performance Evaluation Methodology

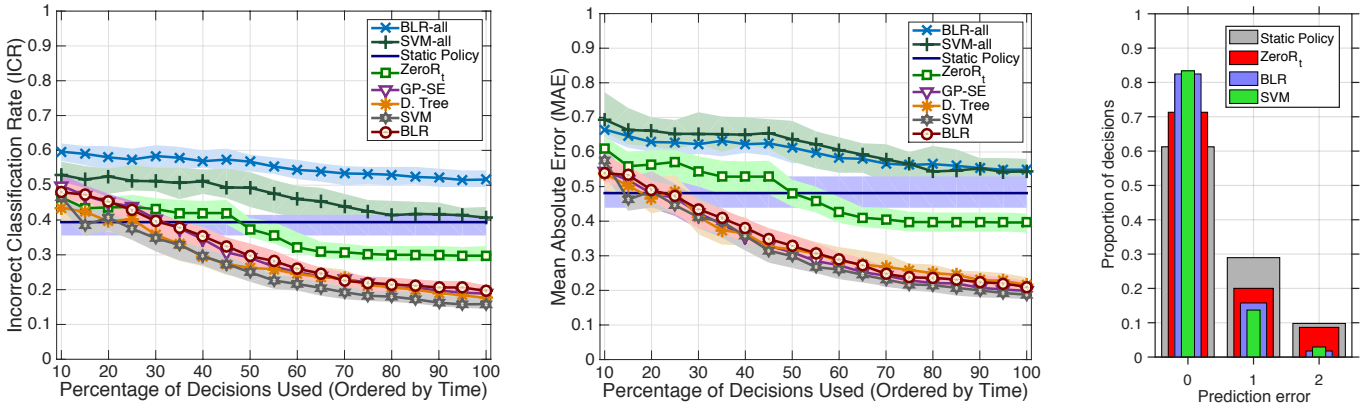
We developed a machine learning framework to reliably estimate the error measure for different methods. Our framework uses the standard splitting of the data into training and testing sets. We randomly select 50% of the participants for testing ($U = 20$) to compute an estimate of the error, and the remaining 21 participants for training to learn the parameters for the BLR model by maximizing the log-likelihood. Note that, for baseline and other methods evaluated, there are no parameters to learn.

On the 20 test participants, we estimate the error measure as follows. We first form datasets $\mathcal{D}_{u,t_u}$ for each participant. As participants have varied number of decisions, we select a fixed percentage of each participant's data, e.g., for each participant, we can select the first 10% of their decisions, that we denote by $t_u = 10\%, \forall u$. We then form the dataset $\mathcal{D}_{test}$ by randomly selecting $N_u = 20$ decisions[7] as test decisions y_{ui} for each participant. Using the method $\mathcal{M}$, we compute predictions $\hat{y}_{ui|\mathcal{D}_{t,u}}$ for all $y_{ui} \in \mathcal{D}_{test}$. We choose only those decisions as test points that were made after the first half of the decisions so that the test decisions resemble decisions that the participant has to make in the far future. We repeat the above splitting process 50 times with different random seeds to get 50 different realizations of the error.

For our evaluation, we use features chosen using an additive approach, also known as Forward Stepwise Selection [52]. We add each of the 37 features (Sections IV-C and V-A6) in turn, and observe their effect on the model's performance. The feature that most improves performance is selected. We

[6] In practice, these thresholds should be learned from the data.

[7] Some participants do not have more than 75 decisions, hence choosing 20 test decisions (around 30% of their data) allow us to include them.



(a) (b) (c)

Fig. 6. Performance results for the machine learning models evaluated in our study. (a) and (b) show estimates of error measures as a function of t_u which is varied from 10% to 100%. (a) is obtained by using the 0-1 loss and reports ICR, while (b) is obtained by using MAE. In each plot, thick lines (with markers on them) show the median of the error, while the shaded region behind shows the error between 25^{th} and 75^{th} percentile. Models that consider contextual information have significantly lower error than our context-oblivious baselines. Moreover, per-user models outperform one-size-fits-all models, i.e., BLR-all and SVM-all. (c) shows the histogram of MAE over test decisions in $\mathcal{D}_{test}$ for one random partition and $t_u = 100\%$. Context-aware methods make very few mistakes with a loss of 2, which clearly shows that such methods rarely make the mistake of predicting "allow" for "deny" and vice versa.

repeat the procedure to find the second most important feature, and so on. We continue this procedure until the performance remains the same or decreases, as shown in Figure 8 for BLR. Using this approach, we selected the following seven features for BLR: method category (i.e., location, contacts, or storage), method name (i.e., the actual API call), app name, whether the app was in the foreground, whether denying the request causes the app to crash, day of month, and battery-level percentage. Note that the effect on performance of a set of features will depend on the machine learning model selected. This is not necessarily the best subset and combination of features for BLR, as our selection approach was not exhaustive; the best subset and combination of features may vary across participants. For BLR, a possible approach is to use regularization to find the best features, which also helps reducing overfitting.

In our evaluation, we use data from all the participants, but only for decisions associated with popular apps, i.e., apps with more than 200 decisions (Figure 5): Facebook, Twitter, Instagram, WhatsApp, Viber, Skype, Snapchat, The Weather Channel, and AccuWeather. We do so because we do not have enough data for the remaining apps to reliably perform our analysis (see Table II in the Appendix).

Our experimental framework and the models evaluated (see next section) were implemented by using the Matlab Statistics and Machine Learning toolbox, and the GPML toolbox [53]. Our code is publicly available in the *SmarPer*'s website [11].

F. Performance Evaluation Results

In our evaluation, we considered the following models: static policy, ZeroR$_t$, BLR, Gaussian Process with Squared Exponential Kernel (GP-SE), decision tree (D. Tree), and 3-binary support vector machines (SVM) with linear kernel. The goal was to compare context-oblivious models with different context-aware models. We also evaluated the training of one-size-fits-all models (i.e., BLR-all and SVM-all), i.e., training a single model for all users.

Figure 6 shows estimates of error measures as a function of t_u. We vary t_u from 10% to 100% for all test participants. Figure 6(a) is obtained by using 0-1 loss function and shows the median of the ICR obtained by the different models evaluated. The shaded area shows the region between 25^{th} and 75^{th} percentile. We can observe that both one-size-fits-all models (i.e., BLR-all and SVM-all) have a significantly higher error rate than most per-user models; BLR-all performs even worse than our baselines. These results are consistent with our observations about Figure 3, participants' unique privacy preferences make it difficult to train a one-size-fits-all model that accurately predicts decisions at runtime. For $t_u = 100\%$, the mean ICR is 0.39 ($\pm$0.04) for static policy, 0.30 ($\pm$0.03) for ZeroR$_t$, 0.20 ($\pm$0.03) for BLR, and 0.16 ($\pm$0.02) for SVM. We can see that context-aware models obtain a much lower error-rate than the baselines, which clearly shows the gain obtained after adding context. Also note that, unlike the static policy method, all the other per-user models are dynamic and *learn* to predict better as the amount of data is increased. In addition, note that BLR, SVM, GP-SE and D. Tree have roughly similar performance. Still, BLR can be considered a safer option, due to its simplicity, i.e., lower risk of overfitting and computational overhead [54].

Figure 6(b) shows a similar trend for MAE loss. For $t_u = 100\%$, the mean MAE was 0.48 ($\pm$0.06) for static policy, 0.39 ($\pm$0.04) for ZeroR$_t$, 0.22 ($\pm$0.03) for BLR, and 0.19 ($\pm$0.03) for SVM. To put these numbers in perspective, note that the MAE is in the range [0,2]. The gains obtained with our context-aware methods are even larger here because the MAE loss function captures the ordering between different types

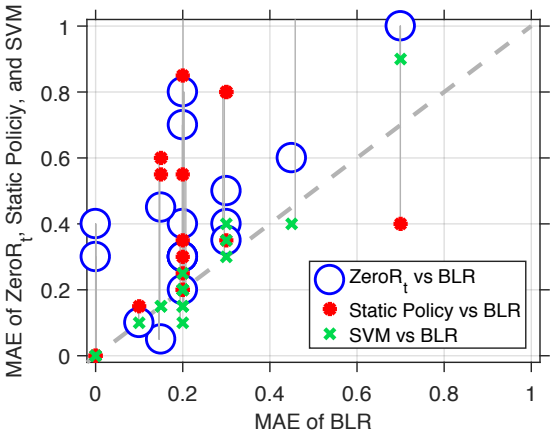


Fig. 7. Comparison of the individual performance on 20 participants for one random partition and $t_u = 100\%$. We plot MAE obtained by the baselines and a SVM versus those obtained by our BLR model. Each point corresponds to the MAE of a participant estimated using (3) with $N_u = 20$ and $U = 1$. Note that not all the points are visible due to the plot's scale. A thin grey line joins the MAEs obtained on the same participant. A point above the dashed grey line indicates that the corresponding baseline gives worse performance than our method, which is the case for most participants. Also, we can see that SVM and BLR have comparable performances.

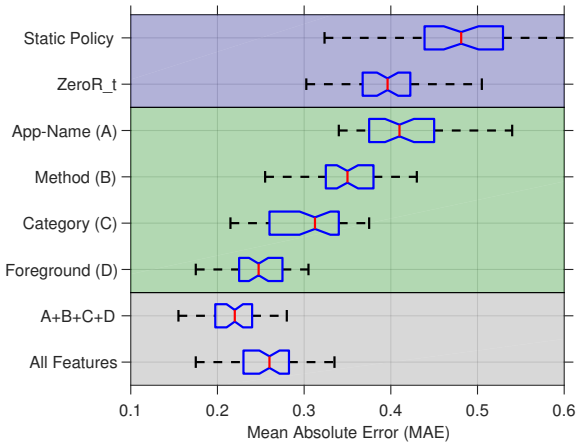


Fig. 8. Importance of individual contextual features for prediction. We show box-plots of MAE obtained for 50 different partitions with $t_u = 100\%$. The top two box-plots are for the baselines, and the next four are for BLR models using only one feature. Even with one feature containing a single context regarding the app being in the foreground or background, BLR outperforms the baselines. The last two box-plots use all four features and all 37 features, respectively, where we see that BLR with the four features performs slightly better. For the details of features, see Sections IV-C and V-A6.

of decisions, which is ignored by the 0-1 loss. For example, MAE penalizes decisions according to the degree of privacy violation, i.e., predicting "allow" for "deny" has a loss of 2 compared to predicting "obfuscate" which has a loss of 1. Under 0-1 loss these errors are treated equally with a loss of 1. Thus, MAE is a better measure of the loss for our problem. Although we believe our results are quite encouraging (more than 80% of correct predictions for modest training set sizes, hence lower user burden), the level of user satisfaction for such values of the performance metric must be evaluated through dedicated experiments and user studies, which will be carried out in the second phase of the project.

Furthermore, we show in Figure 6(c) the distribution of MAE over test decisions in $\mathcal{D}_{test}$ for one random partition with $t_u = 100\%$, i.e., all the training data. Context-aware methods such as BLR and SVM have very few mistakes with a loss of 2, which clearly shows that such methods very rarely make the mistake of predicting "allow" for "deny" and vice versa (see Section VII-C for more details about the impact of such mistakes). Again, there is little difference between the results for BLR and SVM.

The sensitivity of MAE to different types of decisions is also reflected in the worse performance of ZeroR$_t$ in Figure 6(a) compared to its performance in Figure 6(b). ZeroR$_t$ has many predictions with MAE of 2; they are ignored under the 0-1 loss but not under the MAE loss, which is why ZeroR$_t$ performs worse with the latter. In contrast, our context-aware methods perform similarly under both loss functions.

Figure 7 compares the individual performances on 20 participants for one random partition and $t_u = 100\%$. We plot MAE losses obtained by the baselines and SVM versus those obtained by BLR. Each point corresponds to the MAE of a participant estimated using (3) with $N_u = 20$ and $U = 1$. The MAE of the static policy method, ZeroR$_t$ and SVM methods are shown by crosses, circles, and dots, respectively. A thin grey line joins the two MAEs obtained on the same participant. A point above the dashed grey line indicates that the corresponding baseline gives worse performance than BLR, which is the case for most participants. Among the baselines there is no clear winner. Similarly, there is no clear winner between BLR and SVM. Note that these MAE values are less stable as they are estimated with only 20 points. For many participants, standard errors are in the order of 0.1. The numbers reported in Figure 6 are more stable and reliable since they are estimated with a large number of test decisions (20 participants with 20 test decisions each giving us a total of 400 points). Nevertheless, Figure 7 shows that even across participants adding context improves the performance.

Also, note that in Figure 7 the variance across participants is quite high. We can predict some participants very well (MAE is close to zero), whereas for others MAE could be as high as 0.7. The aggregate over participants however is quite satisfactory, e.g., in Figure 6(c) where each participant is equally represented (each participant contributes 20 test decisions in the histogram). For the participants we cannot predict well, possible reasons are that more data is needed or that they were not consistent in their decisions.

Figure 8 explores the importance of individual features for prediction. We show box-plots of MAE obtained for 50 different partitions with $t_u = 100\%$. The top two box-plots are for static policy and ZeroR$_t$, respectively. The next four box-plots show the performance of BLR obtained after adding only one feature out of the following four features: (A) the name of the app requesting permission, (B) the method of the

request, i.e., the actual API call, (C) the method category, i.e., contacts, location, or storage, and (D) whether the requesting app was in the foreground. Even with just one feature, we obtain improvements over the baselines that use (almost) no contextual information at all. The most striking among these features is the feature D (regarding the app being in the foreground or not) which obtains a much lower median error of 0.25 compared to the baselines. The 7^{th} box-plot shows the performance obtained with all the 4 features (A+B+C+D) which achieves a slightly lower and robust MAEs compared to the last box-plot which shows performance when all 37 features are used. This type of behavior is expected when the sample-size is small which is the case here.

G. Computational Performance

Our chosen machine learning model, BLR, is simple enough to run on smartphones. To evaluate its computational performance, we used a Motorola Moto G 3^{rd} generation smartphone with Android 5.1.1, 7 contextual features, 5-fold cross-validation, and around 200 decisions from a single participant. We ported our BLR algorithm to Android using the Efficient Java Matrix Library (EJML) [55]. With this setup, training took around 1.32 ± 0.31 s and prediction 50 ± 6 μs, and the CPU usage was not higher than 50%. These results show that our approach is feasible in smartphones, particularly if we take into account that training does not need to happen frequently (e.g., a couple of times during the day). Moreover, training can be done while the smartphone is idly charging to avoid draining the battery or interfering with apps. In future work, we also plan to evaluate BLR with sequential updating (i.e., online learning), by using one rank update of the Cholesky factor or stochastic gradient descent. Such approach should reduce training time significantly. In addition, we estimated the impact on performance of our *SmarPer* prototype, particularly the service that collects context information and intercepts apps' requests. Using OS and third-party tools, we did not measure a significant impact on CPU usage from our service. Regarding battery life, we used Android's Battery monitor API to measure if using *SmarPer* drained the battery faster, but we did not measure a significant difference on battery life between a smartphone with and without *SmarPer*. Moreover, none of the participants reported problems with battery life.

VII. Discussion

In this section, we present further discussion of the results obtained in our data-collection and machine learning analysis, as well as deployment considerations for *SmarPer*.

A. Amount of Training Data vs Model Complexity

Figures 6(a) and 6(b) show that the MAE and ICR for our BLR model continue to decrease by the end of the experiment, i.e., $t_u = 100\%$. Thus, the performance of our model can be further improved with more data. In contrast, for ZeroR$_t$, they flatten out around $t_u = 80\%$, hence, more decision data may not help improving the performance of ZeroR$_t$.

As there is a wide variance in users' preferences, it is recommended to collect data from a sufficiently large number of users. The number of decisions per user depends on the type of their preferences as well as on the dimensionality of the contextual features. A larger amount of training data will enable the application of advanced machine learning models that can capture the wide variance in the privacy preferences, for example, a topic model can represent a user as a mixture of several types of privacy preferences and is likely to be much more accurate [56]. In short, the amount of training data and time required to train an accurate model depend on the user and their willingness to provide decision data.

In our experiments, we did try several non-linear models that are based on Gaussian process regression (GP-SE), as well as a SVM and decision tree models. These models only marginally improved the performance over a linear model (Figures 6(a) and 6(b)), suggesting that the amount of data is perhaps not enough for training more complex models. On 41 users with 8,521 decisions with 37 contextual features, a linear model worked the best. As a next step, we plan to collect additional decision data and to evaluate more advanced machine learning models to improve prediction accuracy.

B. Automating Permission Decisions

We ran several experiments to evaluate how our BLR model predicts and automates decisions to reduce users' overhead, i.e., the number of prompts to answer. For this purpose, we estimated the confidence of our model on each decision (using the mean and variance) and defined different thresholds to decide whether to automate the decision or prompt the user. However, as stated before, we did not have enough data to reach concrete conclusions. Moreover, we face the challenge of determining when the model is accurate enough to start automating decisions. On the one hand, our model needs as much decision data as possible to improve its performance, but on the other hand, once our model starts automating decisions, it will obtain less decision data.

The simplest approach is to make decision prompts randomly and limit the number of requests per day. In our experiments, this works reasonably, although we found that ultimately using all the data gives the best results. This implies that all of our data are useful for prediction. This also implies that, in practice, an automatic system might collect a few responses each day until enough decisions have been collected.

We could follow the approach used in our data collection phase, i.e., to prompt users a limited number of times per day to collect data to train our model, and to rely on user-defined static policies for other requests. To collect enough decisions per app and reduce the risks of overwhelming the user, we recommend to start collecting data only for a subset of apps selected by the user (e.g., most used apps); gradually, other apps can be added to the system. Another option, is to profit from the data from similar users to accelerate the learning process, i.e., user profiles. However, the system first needs to learn the various types of preferences. Once we have data from a sufficient number of users and apps, we will be able to map a

new user to a particular profile and start automating decisions for certain requests instead of relying on static policies. As mentioned before, our data set does not contain enough users to identify profiles via clustering.

Using our machine learning framework, we could regularly train and test our model to monitor its performance (e.g., once a day). If the MAE is lower than a defined threshold (function $c()$, see Section III-C), the decisions are made automatically, using the learned model. To keep the model updated, we can follow an exploration-exploitation trade-off [57], where we prompt the user for more decision data at a rate that is an increasing function of the current estimation of the MAE (i.e., the higher the estimated error, the higher the prompt rate). Thus, the decision on a request would depend both on the current estimate of the user's preference and on random explorations, i.e., the mean can be used to exploit and the variance to explore. These learning methods are popular in machine learning and are useful for automating decisions.

The acceptable MAE is a user-dependent value and can be estimated only via a long-term study and/or interviews. If the decisions are balanced across allow, obfuscate, and deny, predicting obfuscate all the time would give an MAE of 0.66. We could suggest some reasonable values (e.g., 0.20), but the user must decide which value is acceptable.

Moreover, as discussed in Section III, *SmarPer* includes an audit function for correcting incorrect automatic decisions. Such corrected decisions are added to the training set, potentially with higher weights, to prevent the system from repeating the same errors. This feature was not evaluated in our first study, as no automatic decisions were made during the experiment; but it will be in the second phase of the project. As for the adoption of the audit feature, we observed during the study that participants fixed their own incorrect decisions: For instance, when participants realize that one of their decisions breaks the app functionality, they clear *SmarPer*'s decision cache and restart the app. Such behavior suggests that the audit feature would be used by users.

Overall, our user survey shows that participants are interested in automatic decisions. For instance, 66% of our participants reported that they will trust a system that makes automatic permission decisions on their behalf. In addition, 88% said they would be "very interested" or "interested" to see a feature like *SmarPer* in a new version of Android.

The second phase of the *SmarPer* project, on which we are currently working, will enable us to evaluate our approach and the user perception with respect to automating decisions (accuracy, frequency and adequacy of the prompts, sensitivity to prediction errors, i.e., determining what is an acceptable value for the MAE and the user preferences with respect to undersharing vs. oversharing) and the use of *SmarPer*'s features, such as the audit option.

C. Impact of Predicting Permission Decisions

SmarPer's purpose is to learn and emulate users' privacy behaviors. That is, if a user tends to put her privacy at risk by sharing large amounts of information, the trained model will do the same. Problems arise if the predicted decision does not match the user's intent. First, if the model predicts "allow", instead of "deny", sensitive information will be revealed to apps, i.e., privacy loss due to oversharing. Second, if the model predicts "deny", instead of "allow", the app will not work as the user expects, i.e., utility loss due to undersharing. Third, if the model predicts "obfuscate", instead of "deny" or "allow", some privacy or utility loss occurs, depending on the scenario (data type), i.e., partial-oversharing or partial-undersharing. Note that, for all practical purposes, undersharing (and some partial-undersharing) errors mean that an app will not longer work because it was denied the permissions required for key functionality. In some cases, an app could crash or behave unexpectedly (see Section VII-D). Still, the chance of such problems is low, now that Android supports disabling permissions (Android 6+), as most apps can gracefully handle denied permissions. In short, *SmarPer* should be evaluated with respect to its ability to mimic users' decisions and to the types of incorrect predictions.

As Figure 6 shows, BLR and SVM models rarely make large mistakes predicting "allow" for "deny" and vice versa. More specifically, BLR has an average *per-user* of $0.6\pm0.6\%$ oversharing error, $9.4\pm1.9\%$ partial-oversharing error, $8.8\pm1.8\%$ partial-undersharing error, and $0.8\pm0.5\%$ under-sharing error. In contrast, static policy (currently deployed approach) has an average *per-user* of $7.1\pm2.6\%$ oversharing error, $17.4\pm3.0\%$ partial-oversharing error, $11.5\pm3.2\%$ partial-undersharing error, and $2.6\pm2.0\%$ under-sharing error. As a result, we can see that, compared with static policy, our approach significantly reduces the loss of privacy and utility. Also, note that the errors reported for our approach assume that all the predicted decisions are automated. In practice, *SmarPer* will automate only a subset of the predicted decisions based on the output of the function $c()$ (Section IIC). Hence, the errors should be smaller. To reduce these errors further, we can also use cost-sensitive training [50], where users can configure the kind of errors to minimize (oversharing/undersharing), as in [28].

D. Data Obfuscation

Obfuscation was well-received by the participants. Over our whole data set, 29% of decisions were obfuscate decisions. A similar fraction was observed across data types: Users obfuscated 25% of requests for contacts, 26% of requests for location, and 32% of requests for storage. Moreover, in our exit survey 73% of participants found obfuscation useful and 80% stated that they would like to obfuscate additional data types. Still, some participants did not find obfuscation useful, e.g., a few participants chose to always deny or always accept apps' requests. Also, a small number of participants reported that obfuscation caused problems with certain apps, hence they stopped using this option. It is also possible that, in spite of the training provided, some participants did not fully understand the purpose of obfuscation.

In our exit interviews, participants shared some use cases for obfuscation: *"For apps that need location, such as Accuweather, I was giving obfuscated access most of the time*

as I expected the functionality to be fine with coarse grained location."–D69. Another participant explained the benefits of obfuscation: *"Even if we don't want to give all the information to the application used, we still have to give some, in order to the application to be useful"–37F*. See Table III (Appendix B).

Obfuscation may introduce unexpected behaviors in apps [24]. In our field test, few participants reported non-critical issues with obfuscation. For instance, some participants reported that storage obfuscation interfered with apps that take or edit photos (e.g., Twitter, Facebook), as they need to access the pictures folder. Also, participants reported that WhatsApp was not displaying contact name's correctly when obfuscation was chosen for contacts. Coordination with developers and mobile platform providers is important to reduce these problems. For example, native APIs could be introduced to handle obfuscated data types and handle possible runtime exceptions.

E. Privacy Benefits of a Per-User Model

Prior works aggregate permission preferences and related information from all users to train a one-size-fits-all classifier [2], [8] or to identify privacy profiles via clustering [8], [9]. However, aggregating this information introduces privacy risks, as it can be misused to infer sensitive information about users. In *SmarPer*, we show that is feasible to train a model per user directly in the user's smartphone, i.e., no permission information is sent to other parties. Note that in the case of BLR, partial decision data from a subset of users is needed to learn the model hyperparameters. However, instead of sending raw decision data to a central location, users can send sufficient statistics (e.g., expectation of β_u^0 and $\boldsymbol{\beta}_u^T$) to defend against inference attacks. More advanced models (e.g., SVMs) do not need this (but might require more data for training).

F. Limitations

There are many challenges associated with predicting and automating permission decisions at runtime using contextual information. To provide some guidance to future works in these area, we present the main limitations of our approach:

- *Participant's bias*. Due to the nature of our evaluation, participants might have some bias towards a more privacy-preserving behavior and, in particular, towards using obfuscation. Such bias is common in the evaluation of privacy tools and is difficult to avoid. To reduce it, we presented in a neutral way the decision options (allow, obfuscate, deny) to participants during their training.
- *Focus on popular apps*. To obtain enough decision data per app for our study, we collected data only from popular apps. This might have introduced a bias, as participants are more familiar with these apps and, in some cases, less willing to deny their requests. Yet, collecting data from a larger set of apps requires a long-term user study to collect enough data from less frequently used apps. Also, techniques to deal with data sparsity will be needed.
- *Simplified decision modeling*. Our prediction model takes into account only contextual factors that can be collected by the users' smartphones. There are probably other contextual factors that are also important but not considered in our approach. Moreover, there are non-contextual factors that are important in privacy decision-making, e.g., psychological factors [20].
- *Data set size.* Our data set is not large enough to reliably train advanced machine learning models, to cluster participants, and to obtain a reliable confidence metric. Hence, we will carry out a longer-term data collection campaign with more participants.
- *Data quality.* We took measures to remove noisy decision data from our data set (Section V). However, we cannot fully validate if participants made decisions that correctly conveyed their intentions. For instance, a participant might have provided only biased or fake decisions, or just random decisions. This could explain why some participants are more difficult to predict.

VIII. Conclusion

As the number of apps, data types, and permissions increases in mobile platforms, users are increasingly in need of better mechanisms to manage permissions. Artificial intelligence and machine learning techniques are already being used for security purposes in mobile platforms such as Android [58]. Therefore, it is logical to use similar techniques to help users to better control their privacy in mobile devices.

We presented *SmarPer*, an advanced permission mechanism for Android, designed to address two important limitations of current mobile permission systems: the static nature of current permission policies and the poor trade-off between privacy and utility. After seeing some initial training decision data and its corresponding contextual information, *SmarPer* uses machine learning to infer users' permission-decision patterns at runtime and to automatically respond to future permission requests (not yet supported). Furthermore, *SmarPer* offers "obfuscated" decisions to reduce the information revealed to apps while still obtaining some utility.

We deployed, under realistic conditions, our *SmarPer* implementation to 41 users and collected their permission decisions and contextual information for around 10 days each. Using a Bayesian linear regression approach to train a model per user, we achieved a mean incorrect classification rate (ICR) of 0.20 ($\pm$0.03), which is a mean relative improvement of 50% over a static policy baseline, i.e., the approach used by current permission systems. Our results show that our model can learn to predict users' decisions with good accuracy and that contextual information is important for such a task.

For future work, we plan to expand our data set of permission decisions to train more advanced models for better accuracy. Moreover, we will evaluate exploration-exploitation trade-off methods to automate decisions at runtime to complete our *SmarPer* implementation and to move a step closer towards automatic permission-management in smartphones.

REFERENCES

[1] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao, "Understanding and Capturing People's Privacy Policies in a Mobile Social Networking Application," *Personal Ubiquitous Computing*, vol. 13, no. 6, pp. 401–412, 2009.

[2] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android Permissions Remystified: A Field Study on Contextual Integrity," in *Proceedings of the Usenix Security Symposium*, 2015.

[3] "CyanogenMod," https://en.wikipedia.org/wiki/CyanogenMod, Last visited: February 2017.

[4] M. Bokhorst, "XPrivacy - The ultimate, yet easy to use, privacy manager," https://github.com/M66B/XPrivacy, Last visited: February 2017.

[5] "LBE Privacy Guard," http://forum.xda-developers.com/showthread.php?t=1091065, Last visited: January 2017.

[6] P. Sawers, "Android users have an average of 95 apps installed on their phones, according to Yahoo Aviate data," http://thenextweb.com/apps/2014/08/26/android-users-average-95-apps-installed-phones-according-yahoo-aviate-data/, Last visited: February 2017.

[7] K. Olmstead and M. Atkinson, "Apps Permissions in the Google Play Store," http://www.pewinternet.org/2015/11/10/apps-permissions-in-the-google-play-store/, Last visited: February 2017.

[8] B. Liu, J. Lin, and N. Sadeh, "Reconciling Mobile App Privacy and Usability on Smartphones: Could User Privacy Profiles Help?" in *Proceedings of the International Conference on World Wide Web (WWW)*, 2014.

[9] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. Zhang, N. Sadeh, A. Acquisti, and Y. Agarwal, "Follow My Recommendations : A Personalized Privacy Assistant for Mobile App Permissions," in *Proceedings of the Symposium On Usable Privacy and Security (SOUPS)*, 2016.

[10] F. Shih, I. Liccardi, and D. J. Weitzner, "Privacy Tipping Points in Smartphones Privacy Preferences," in *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, 2015.

[11] "SmarPer Project: Automatic and Context-Aware Permissions for Android," https://spism.epfl.ch/smarper/, Last visited: March 2017.

[12] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," in *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, 2012.

[13] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as Part of the App Decision-Making Process," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2013.

[14] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) Permissions Used by Android Apps," in *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, 2013.

[15] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to Ask for Permission," in *Proceedings of the USENIX Workshop on Hot Topics in Security (HotSec)*, 2012.

[16] S. Bugiel, S. Heuser, and A.-R. Sadeghi, "Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies," in *Proceedings of the USENIX Security Symposium*, 2013.

[17] S. Heuser, A. Nadkarni, W. Enck, and A.-R. Sadeghi, "ASM: A Programmable Interface for Extending Android Security," in *Proceedings of the USENIX Security Symposium*, 2014.

[18] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming Information-stealing Smartphone Applications (on Android)," in *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*, ser. Lecture Notes in Computer Science, 2011, vol. 6740, pp. 93–107.

[19] Y. Agarwal and M. Hall, "ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2013.

[20] C. Dong, H. Jin, and B. P. Knijnenburg, "Predicting Privacy Behavior on Online Social Networks," in *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM)*, 2015.

[21] M. Conti, V. T. N. Nguyen, and B. Crispo, "CRePE: Context-Related Policy Enforcement for Android," in *Information Security: 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers*. Springer Berlin Heidelberg, 2011, pp. 331–345.

[22] S. Chakraborty, C. Shen, K. R. Raghavan, Y. Shoukry, M. Miller, and M. B. Srivastava, "ipShield : A Framework For Enforcing Context-Aware Privacy," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[23] M. Miettinen, S. Heuser, W. Kronz, A.-R. Sadeghi, and N. Asokan, "Conxsense: Automated context classification for context-aware access control," in *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, ser. ASIA CCS '14, 2014.

[24] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These Aren't the Droids You're Looking for: Retrofitting Android to Protect Data from Imperious Applications," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2011.

[25] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "MockDroid: Trading Privacy for Application Functionality on Smartphones," in *Proceedings of the Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2011.

[26] J. Cranshaw, J. Mugan, and N. Sadeh, "User-Controllable Learning of Location Privacy Policies with Gaussian Mixture Models," in *AAAI Conference on Artificial Intelligence*, 2011.

[27] I. Bilogrevic, K. Huguenin, B. Agir, M. Jadliwala, and J.-P. Hubaux, "Adaptive Information-Sharing for Privacy-Aware Mobile Social Networks," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (Ubicomp)*, 2013.

[28] I. Bilogrevic, K. Huguenin, B. Agir, M. Jadliwala, M. Gazaki, and J.-P. Hubaux, "A machine-learning based approach to privacy-aware information-sharing in mobile social networks," *Pervasive and Mobile Computing*, vol. 25, pp. 125–142, 2016.

[29] L. Yuan, J. R. Theytaz, and T. Ebrahimi, "Context-Dependent Privacy-Aware Photo Sharing based on Machine Learning," in *Proceedings of 32nd International Conference on ICT Systems Security and Privacy Protection (IFIP SEC 2017)*, 2017.

[30] J. Lin, B. Liu, N. Sadeh, and J. I. Hong, "Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings," in *Proceedings of the Symposium On Usable Privacy and Security (SOUPS)*, 2014.

[31] rovo89, "Xposed Installer — Xposed Module Repository," http://repo.xposed.info/module/de.robv.android.xposed.installer, Last visited: January 2017.

[32] ——, "Xposed Development Tutorial," https://github.com/rovo89/XposedBridge/wiki/Development-tutorial, last visited: January 2017.

[33] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. Le Boudec, "Protecting Location Privacy: Optimal Strategy Against Localization Attacks," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[34] N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Optimal Geo-Indistinguishable Mechanisms for Location Privacy," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[35] A. T. Truong, Q. C. Truong, and T. K. Dang, "An adaptive grid-based approach to location privacy preservation," in *Advances in Intelligent Information and Database Systems*, 2010.

[36] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J. Hubaux, "Quantifying Location Privacy," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2011.

[37] Y. Matsuo, N. Okazaki, K. Izumi, Y. Nakamura, T. Nishimura, and K. Hasida, "Inferring Long-term User Properties based on Users' Location History," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2007.

[38] A. Noulas, M. Musolesi, M. Pontil, and C. Mascolo, "Inferring interests from mobility and social interactions," in *Proceedings of Workshop on Analyzing Networks and Learning with Graphs (NIPS'09)*, 2009.

[39] I. Panagiotis, I. Polakis, E. Athanasopoulos, F. Maggi, and S. Ioannidis, "Face/Off: Preventing Privacy Leakage From Photos in Social Networks," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2015.

[40] S. S. Jon Howell, "What You See is What They Get: Protecting users from unwanted use of microphones, cameras, and other sensors," in *Proceedings of the Web 2.0 Security and Privacy Workshop*, 2010.

[41] J. Lemley, S. Abdul-Wahid, D. Banik, and R. Andonie, "Comparison of recent machine learning techniques for gender recognition from facial images," in *Proceedings of the Modern Artificial Intelligence and Cognitive Science Conference*, 2016.

[42] X. Lu and A. Jain, "Ethnicity identification from face images," in *Proceedings of SPIE - The International Society for Optical Engineering*, 2004, vol. 5404, pp. 114–123.
[43] N. K. Malhotra, S. S. Kim, and J. Agarwal, "Internet Users' Information Privacy Concerns (IUIPC): The Construct, the Scale, and a Causal Model," *Information Systems Research*, vol. 15, no. 4, pp. 336–355, 2004.
[44] S. Garavaglia and A. Sharma, "A Smart Guide to Dummy Variables: Four Applications and a Macro," in *Proceedings of the Northeast SAS Users Group Conference*, 1998.
[45] P. A. Norberg, D. R. Horne, and D. A. Horne, "The Privacy Paradox: Personal Information Disclosure Intentions Versus Behaviors," *Journal of Consumer Affairs*, vol. 41, no. 1, pp. 100–126, 2007.
[46] "ZeroR," https://weka.wikispaces.com/ZeroR, Last visited: February 2017.
[47] K. E. Train, *Discrete Choice Methods With Simulation*, Cambridge: Cambridge University Press, Ed., 2009.
[48] C. K. Williams and C. E. Rasmussen, "Gaussian Processes for Regression," in *Advances in Neural Information Processing Systems (NIPS)*, 1995.
[49] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
[50] J. A. Anderson, "Regression and Ordered Categorical Variables," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 46, no. 1, pp. 1–30, 1984.
[51] Z.-H. Zhou and X.-Y. Liu, "On Multi-Class Cost-Sensitive Learning," in *AAAI Conference on Artificial Intelligence*, 2006.
[52] T. H. R. T. Gareth James, Daniela Witten, *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.
[53] "Gaussian Process Regression and Classification (GPML) Toolbox," http://www.gaussianprocess.org/gpml/code/matlab/doc/, Last visited: February 2017.
[54] "How to choose algorithms for Microsoft Azure Machine Learning," https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-choice, Last visited: February 2017.
[55] "Efficient Java Matrix Library (EJML)," http://ejml.org/wiki/index.php?title=Main_Page, Last visited: February 2017.
[56] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.
[57] E. Brochu, V. M. Cora, and N. De Freitas, "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning," *arXiv preprint arXiv:1012.2599*, 2010.
[58] "Google's Training Its AI to Be Android's Security Guard," http://www.wired.com/2016/06/googles-android-security-team-turns-machine-learning/, Last visited: February 2017.

ACKNOWLEDGMENT

This work was partially funded by the Swiss National Science Foundation with grant 200021-138089 (PICAWA). We would like express our sincere gratitude to Ketevani Zaridze and John Stephan for their contributions to this project. The authors are also extremely grateful to the anonymous reviewers and to the shepherd, Lujo Bauer, for their insightful comments and guidance.

APPENDIX A
DISTRIBUTION OF DECISIONS PER APP

Table II shows the total number of decisions per app collected during our study (Section V). For each app, it also shows the distribution of decisions per data type (i.e., contacts, location, storage) and for each data type the distribution of the type of decision (i.e., allow, obfuscate, deny). These numbers demonstrate the importance of selecting popular apps in our study to be able to collect enough decision data in the allotted time; only 9 apps have more than 200 decisions. Hence, for less popular apps or apps that are used less frequent, we need longer periods of time to collect enough data (avoiding forcing participants to generate decisions to reduce bias).

APPENDIX B
EXAMPLE SCENARIOS USED IN PERMISSION DECISIONS

After completing our data collection campaign, we sent an e-mail to some of the participants with a questionnaire inquiring about example situations in which they chose a decision for a particular app and data type. Some of the answers collected are presented in Table III. These responses show the trade-off between privacy and usability that users face when making permission decisions. These responses also provide an idea of the scenarios or contexts that users consider when making decisions.

		Contacts				Location				Storage			
App	**Total**	**Count**	**A**	**O**	**D**	**Count**	**A**	**O**	**D**	**Count**	**A**	**O**	**D**
WhatsApp	2982	1246	871	257	118	460	103	105	252	1276	765	223	288
Facebook	1821	18	4	13	1	1007	180	186	641	796	171	269	356
The Weather Channel	878	0	0	0	0	311	120	112	79	567	66	170	331
Twitter	569	1	1	0	0	203	23	72	108	365	186	119	60
Viber	556	202	87	93	22	115	61	42	12	239	62	150	27
Skype	431	87	55	16	16	152	13	60	79	192	19	87	86
Instagram	350	0	0	0	0	44	25	6	13	306	103	88	115
AccuWeather	283	0	0	0	0	233	100	109	24	50	5	23	22
Snapchat	205	32	13	5	14	59	1	16	42	114	60	27	27
TripAdvisor	153	0	0	0	0	81	34	29	18	72	21	31	20
Shazam	93	0	0	0	0	50	35	11	4	43	10	30	3
Dropbox	56	0	0	0	0	0	0	0	0	56	11	39	6
Evernote	45	0	0	0	0	7	0	1	6	38	1	30	7
Waze	36	11	0	0	11	13	2	6	5	12	0	7	5
iHeartRadio	24	0	0	0	0	17	0	4	13	7	0	2	5
SoundCloud	16	0	0	0	0	0	0	0	0	16	5	9	2
Runtastic	11	0	0	0	0	5	0	1	4	6	0	2	4
Uber	6	1	0	1	0	4	2	1	1	1	0	1	0
Heroes	2	0	0	0	0	0	0	0	0	2	1	0	1
Subway Surf	2	0	0	0	0	0	0	0	0	2	0	2	0
Wish	1	0	0	0	0	0	0	0	0	1	0	1	0
Yelp	1	0	0	0	0	0	0	0	0	1	0	0	1

TABLE II

DISTRIBUTION OF THE PERMISSION DECISIONS PER APP, TYPE OF DATA (I.E., CONTACTS, LOCATION, STORAGE) AND TYPE OF DECISION (I.E., ALLOW, OBFUSCATE, DENY).

App	**Method category**	**Decision**	**Example Situations**
WhatsApp	Contacts	Allow	(1) "Adding new contacts."; (2) "Used because otherwise the names of the contacts within the app were missing, it wasn't really convenient otherwise."
		Obfuscate	(1) "I would use the obfuscation of the contacts most of the times, because it gives the required information to Whatsapp (phone number for example) while protecting my privacy."
		Deny	(1) "Never, because I thought that without contacts, Whatsapp is useless."
WhatsApp	Location	Allow	(1) "Only selected when the pop-up was red." (when it may cause the app to crash)
		Obfuscate	(1) "For finding out position of my friends while chatting."
		Deny	(1) "I denied the access unless I wanted to use a feature of the applications that required my location, like sharing my position with a friend within WhatsApp."
WhatsApp	Storage	Allow	(1) "To upload some picture."
		Obfuscate	(1) "Used most of the times, so that if the app needs to access some content that is in its folder."
		Deny	(1) "When not actively using the app."
TripAdvisor	Location	Allow	(1) "When I wanted to see my exact position relative to a bar or a restaurant."
		Obfuscate	(1) "When I wanted to see all the restaurants available near my house, but do not wanted to give the precise location where I live."; (2) "When we do not want to give our exact location, but still need a service depending on our location."
		Deny	(1) "When I did not need location services."
AccuWeather	Location	Obfuscate	(1) "I used it most of the time as I expected the functionality to be fine with coarse grained location."

TABLE III

EXAMPLES OF REPORTED SITUATIONS WHERE PARTICIPANTS MADE A PERMISSION DECISION FOR A PARTICULAR APP AND DATA TYPE.