MoRePriv: Mobile OS Support for Application Personalization and Privacy

Drew Davidson
University of Wisconsin

Benjamin Livshits
Microsoft Research

Abstract

This paper advocates for operating system support for personalization and describes MoRePriv, an operating system *service* implemented on top of the Windows Phone operating system. The approach presented in this paper combines the frequently conflicting goals of *privacy* and *content personalization* on mobile devices. We argue that personalization support should be as ubiquitous as location support, and should be provided by the OS instead of apps.

To enable easy application personalization or *skinning*, MoRePriv approximates users' interests using *personas* such as technophile or business executive. We demonstrate how always-on user interest mining can effectively and accurately infer user interests in a mobile operating system by parsing and classifying multiple streams of (sensitive) information about the user within the OS, such as their email, SMS, Facebook stream, and network communications. For privacy protection, this sensitive information is distilled to a coarse-grained profile, without being exposed to apps, which limits the potential for information leaks.

We show that MoRePriv enables simple, but effective OS-wide *universal personalization*: for example, long drop-down lists in application UIs are *automatically* sorted to better fit the order of users' likely preferences. However, the real power of MoRePriv comes from exposing a personalization API to apps.

Using a number of cases studies, we illustrate how more complex personalization and app skinning tasks can be achieved with the help of MoRePriv. We also argue for better OS support for *ad libraries*, advocating that a more privacy-aware design is possible for mobile advertising, combined with insight into users' preferences and tastes gained with MoRePriv. This approach combines the capabilities of today's powerful ad libraries with privacy concerns of the application, while reducing application permissions and enabling more powerful monetization models. Our experiments show that we are able to reduce app permissions in about 73% of apps that use ad libraries. The ad library study also shows that *removing user tracking* capabilities while providing persona information creates a useful compromise in practice.

1. Introduction

Mobile applications are becoming increasingly *personalized*, fluidly adapting themselves to the needs and preferences of their users. The most common type of personalization most users have encountered is *location-based personalization*: search engine results and Yelp restaurant recommendations are simply better and more relevant when the app knows about the user's location. Other examples of mobile personalization are customizable news delivery apps such as AOL Editions (`editions.com`) and Flipboard (`flipboard.com`). Restaurant and music recommendation services get better with user interaction and training, and mobile butlers such as Apple's Siri and Google's Alfred adapt themselves to the user over time. We believe that this is just beginning.

Much work in personalization has taken the form of aggregating user data in the cloud, then using it for large-scale data mining. This has significant advantages for data aggregators, as it allows them to target services and ads. However, as the recent DoNotTrack debate has illustrated, there are significant downsides for the user. Recently, a series of papers has highlighted the severity of data leak problems for location and other data in mobile apps [7, 9, 15, 24]. In contrast, MoRePriv advocates leaving user data on the mobile device, under the control of the user. The benefits of this simple shift in perspective are many: the user retain control over data in this model and the cloud providers do not have to worry about properly maintaining data, preventing unauthorized data access, complying with local and international laws, running and powering expensive data centers, dealing with PR repercussions of data leaks and unauthorized tracking (such as the recent Apple location data scandal), etc. Also note that MoRePriv, at its core, is an *enabling* technology — it creates extra value and opportunity for all parties. As such, MoRePriv tries to shepherd the ecosystem into a new, more open and productive direction, but it can, if desired, be combined with various privacy modes to eliminate existing privacy leaks [15].

In this paper we present MoRePriv, an operating system service that allows for rich personalization by performing accurate local data collection on the handset without the need to exfiltrate user data from the

handset. Moreover, when user preference information is collected from all apps and OS interactions, it is considerably more accurate compared to what any given app, even the mobile browser, can obtain. We argue that it is to the benefit of users and application developers to draw this information from a single unified and trustworthy source provided by the mobile OS. Personalization functionality can and should be built into the mobile OS and exposed to app developers via an easy-to-use API.

To enable easy application personalization or skinning, MoRePriv approximates user's interests using *personas* such as technophile or business executive. We demonstrate how always-on user interest mining can effectively and accurately infer user interests in a mobile operating system. MoRePriv inference is based on parsing and classifying multiple streams of (sensitive) information about the user within the OS, such as their email, SMS, Facebook stream, and network communications. For privacy protection, this sensitive information is distilled to a coarse-grained persona (as opposed to fine-grained location data, whose leaks have caused much chagrin among users and privacy advocates), without being exposed to apps. The use of personas limits the potential for user tracking: while persona information can be shared by apps to perform server-based personalization, for instance, it is not enough to link the user across multiple interactions.

Just like location services, which are now ubiquitous on most mobile platforms, we believe MoRePriv can unleash developer creativity. Moreover, there is a real chance to create positive change in the mobile ecosystem: as on the web, mobile apps are increasingly ad-funded. Ad targeting necessitates user profiling, which we argue is better delivered through a controlled MoRePriv-based approach, than through privacy-violating tracking. We believe that MoRePriv is timely and can lead to game-changing innovation: while the web tracking infrastructure is quite ingrained, mobile development is less so.

1.1 Mobile Personalization: An Example

Figure 1 demonstrates the power of mobile personalization. A traditional challenge on mobile devices is the lack of screen real estate. As such, long news articles are difficult to read, giving rise to various bookmarking services such as Instapaper, ReadItLater, etc. However, in a news article such as might be found in Wall Street Journal or The New York Times, often only a fraction of the article is relevant to a given user. As such, a customized *summarization strategy* will go a long way toward making the user more productive [21].

Figure 1a shows a text sample, a summarized version of the text with unnecessary details faded 1b, and a excerpt with highlighted entities 1c. While many users will

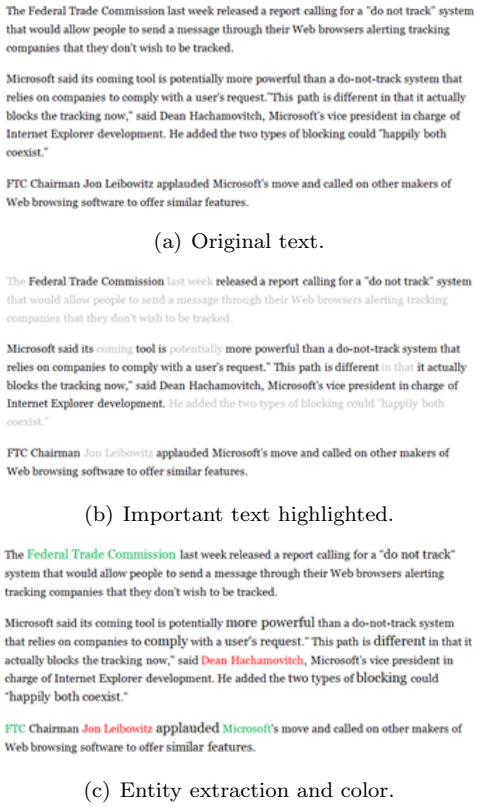


Figure 1. Mobile text view.

prefer versions (b) and (c) to version (a), because both draw attention to important parts of the text, this raises an important question: what *is* important and should be brought to user's attention? The answer is subjective: if the user is someone interested in business or public policy, the aspects of the text that have to do with the FTC and its chairman are relevant. For someone interested in technology, the parts of the text talking about Microsoft are important. This motivates our notion of *personas* such as business executive or technophile, prototypical users with an easily recognizable set of interests that offer targets for *personalization*.

This example of text display illustrates an important general point: many tasks require knowledge of the user to perform well. This has been observed before: most spoken word recognition, dictation, and writing recognition systems work considerably better with training. Of course, having to train the system is a chore for the user. In MoRePriv, training is done at the level of the operating system and draws from other OS functionality as well as user-level apps. This way, the app developer can dedicate their energy to developing better personalization and the user does not have to waste their time training every app they have.

1.2 Contributions

In this paper, we

- propose OS-wide personalization as a way to combine the goals of personalization and privacy on mobile devices, while illustrating key opportunities for mobile operating system designers and propose specific ways in which applications can be enhanced to become more personalized;
- describe our implementation of MoRePriv on the Windows Phone 7.5 OS; and advocate the use of personas, through case studies and end-user experiments;
- demonstrate how to do universal personalization of UI and how MoRePriv APIs can be used for personalization and skinning though several case studies;
- advocate for an OS-supported mobile ad delivery architecture that uses MoRePriv, which has significant advantages compared to the current state-of-the-art in mobile advertising and monetization. In the context of ad delivery and ad libraries, we show how MoRePriv can be *combined* with a privacy-enhancing technology designed to limit the capabilities of mobile ads, while still providing the ability to do ad targeting based on persona data.

1.3 Paper Organization

The rest of this paper is organized as follows: Section 2 provides the background for mobile privacy and personalization. Section 3 gives an overview of MoRePriv. Section 4 describes how we implemented MoRePriv by modifying the Windows Phone operating system and demonstrates the MoRePriv API. Section 5 illustrates the benefits of mobile personalization through several case studies. Section 6 describes how MoRePriv enables an alternative design for mobile ad delivery. Finally, Section 8 concludes the paper. Our companion technical report [2] provides more details on mobile personalization, monetizing mobile apps and restricting advertising, etc.

2. Mobile Personalization: Background

The prevalence of smart phones and other mobile computing devices has opened up new avenues for personalized applications. Because users carry these devices with them wherever they go, they are subject to a level of user interaction never before seen on a personal computing platform. Furthermore, they are typically equipped with cameras, microphones, GPS, and several forms of wireless networking, providing a constant stream of data from both the physical world and the Internet. Common examples of personalization include Siri, the mobile voice assistant and personalized local search. Both, unfortunately, require moving a considerably amount of user data into the cloud for processing. We start out by presenting representative examples of mobile personalization and then proceed to generalize the requirements for a successful personalization platform, such as MoRePriv.

Siri: A familiar example of mobile personalization is the Siri personal assistant, which spun off from a large artificial intelligence undertaking by DARPA. Siri attempts to parse human language in order to perform high-level tasks on behalf of the user, such as making dinner reservations, selecting movies, and making wine recommendations. Over time, Siri learns the user's preferences and personalizes its interface and recommendations accordingly, although the mechanism by which this happens is proprietary. If the user tends to invoke Siri frequently, then this information will pertain to a sizeable portion of the user's day-to-day activities, giving Siri a valuable window into the user's life when targeting content and advertisements. Siri, however, transmits user's utterances to Apple and potentially third parties affiliated with Apple for analysis. As such, both the information given directly by the user such as "set a birthday reminder" and secondary information such as gender and age can be obtained through voice analysis.

GetGlue: GetGlue is a mobile application based on personalization (`getglue.com`). As users consume media and purchase goods, GetGlue allows them to "check in" to these entities. They can then post reviews, ratings, and recommendations, which are tied to their personal profile. GetGlue uses this information to recommend products and content to the user in future interactions, based on the activity of the pool of users in the GetGlue community. GetGlue is partnered with a collection of online merchants and services, who can benefit from the rich data provided by the user base to better target content towards.

Shopkick: Other mobile apps, such as Shopkick (`shopkick.com`), have integrated location-awareness with user preference data. When a Shopkick user enters a brick-and-mortar store, the app takes note of the fact, and consults the user's preferences to offer discounts and recommendations. Users can scan the barcodes of items in which they are interested as they shop, and Shopkick incentivizes this type of data acquisition with payments in a proprietary currency that can be used towards further discounts in the store. This use of personalization is interesting in that it offers traditional merchants the ability to target content, advertisements, and discounts towards users in much the same way as online retailers.

AOL Editions: Editions (`editions.com`) is a mobile news magazine for tablets that learns over time, based on both explicitly stated user preferences and the magazine observing the user clicking on articles. Editions uses a curated taxonomy of news topics, with the user

being actively encouraged to update their interests, creating a virtuous circle of increasingly relevant news content. Editions combines user's calendar within the UI, further adding to the personalized appearance. Other entires in this increasingly busy space are Zite, Pulse, Apollo News, Flipboard, and Early Edition.

Shopitize: Mobile offer app Shopitize (`shopitize.com`) promises to give the user personalized offers based on their shopping receipts, which the user scans into the application. It gives the user a convenient spending report by parsing scanned receipts. This is a good example of the *cost of free*: in exchange for this service, the user provides valuable information that can be used to target them with future offers and services.

3. Overview

The apps above share the following set of personalization steps [19]:

1. Acquire personalization signal from user interactions;
2. Address the *cold start* problem of not having data to base personalization on; often, this is resolved through the use of a secondary source. This is why many modern apps encourage the user to sign up with their Facebook credentials, so that their Facebook data can be "scraped".
3. Refine personalization signal as a result of subsequent interactions.

Opportunities in the Mobile Space: There are several important distinctions for personalization in the mobile space as opposed to desktop computing:

- Data on mobile devices is often more sensitive (or "toxic") than on desktop computers, frequently including personal text messages, phone call records, etc.
- The grip of the ad-based monetization model is not as strong as it is on the web, thus permitting game-changing innovation.
- Finally, processing power on mobile devices is less than on desktop computers, while the energy efficiency requirements are higher.

The majority of research in the mobile personalization space focuses on privacy-preserving ad targeting [3, 11, 12, 30]. While we address mobile advertising in Section 6, we should emphasize that the reach of MoRePriv to enhance the mobile platform is considerably broader, as demonstrated by the case studies in Section 5.

3.1 System Architecture & Design Philosophy

As shown in Figure 2, we envision a system in which both applications and the underlying operating system expose opportunities for personalization. User interactions are observed by *personal preference miners* ("miners", for short), shown at the top of the figure and compiled to a user interest profile, which is subsequently used by a variety of *personalizers* for different forms of personalization, at the bottom of the figure. Personal preference miners are as diverse as location information miners that can tell us about whether the user is around home or on a trip, or miners that discover the activity of the user (walking, riding a train, in a car, etc.) They also pore over user's email, SMS, and Facebook streams to establish user's interests and preferences for news and entertainment.

Survey takers	179	
Concerned about privacy?	149	83%
Use mobile phones?	178	99%
Use smart phones?	154	86%
Install mobile apps?	147	82%
Do you find your persona accurate?	143	77%
For 123 users who use smart phones and install apps and also find the persona accurate		
Would be willing to share (pick one)		
fine-grained location	33	27%
coarse-grained location	35	28%
persona	**55**	**45%**
If it meant more personalized experience, would be willing to share (pick one)		
fine-grained location	40	33%
coarse-grained location	35	28%
persona	**48**	**39%**

Figure 3. Survey result summary.

3.2 User Interest Profile: Personas

Personas are custom representations of various walks of life, offering different targets for personalization. Personas have been advocated before, as a user modeling approach [28]. In addition to being easy to understand, the advatage of this technique is the degree of *pseudonymity* that it provides [20]. In other words, using a persona provides a way to declassify sensitive information; persona data is all that is released by the user to the application, as opposed to personally-identifiable information such as their name or unique identifiers such as the device IMEI.

In our prototype implementation of MoRePriv, we target *eight* personas, shown within the MoRePriv UI in Figure 4(b). Each persona is represented by a Bayesian classifier C_p, trained on a manually curated list of keywords characteristic to profile p. For example, the executive persona represents strong interest in business, finance, and national news. Thus, the corresponding classifier is populated with text from such sites as the Financial Times. On the other hand, the technophile profile represents a strong interest in technology, so the corresponding classifier is populated with text from tech blogs.

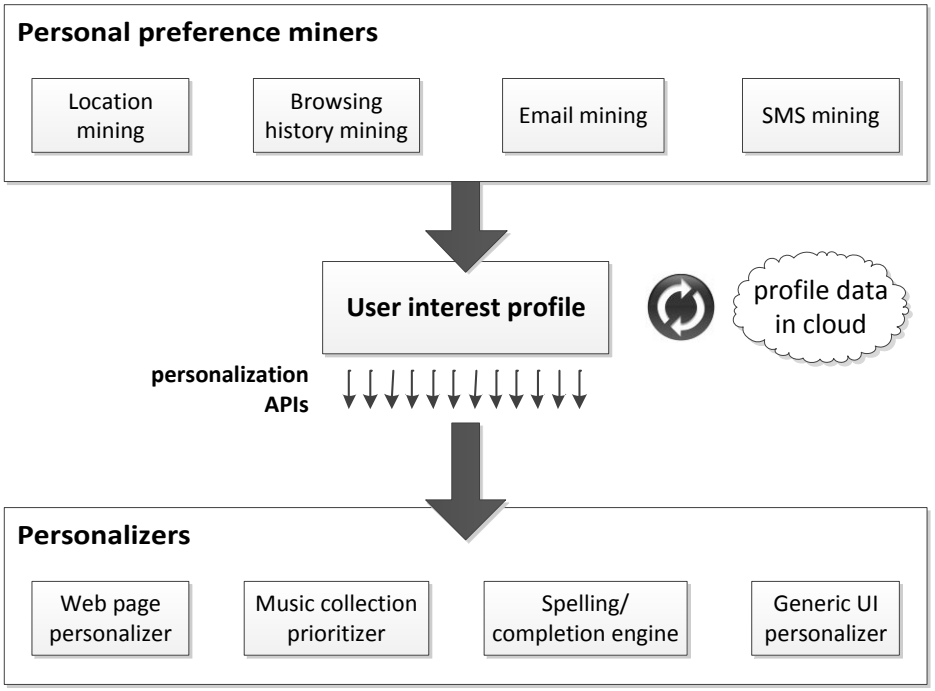


Figure 2. Architecture of MoRePriv.

Our goal in choosing the set of personas is neither completeness nor exclusivity: arguably, it is very difficult to be comprehensive; similarly, there may be intersections between these personas. Instead we optimize for ease of understanding by both the user and the developer. While we believe these profiles are a reasonable proof of concept, we note that our system is modular with respect to the profiles that are used, and the system could easily by modified by training a Bayesian classifier on a new list of keywords.

In practice, no user is likely to have interests that match exactly one persona. As such, each persona is assigned a *persona weight* that indicates how closely that persona matches the user. Consider an example user who is very interested in technology news and somewhat interested in financial news. This user would have a a high persona weight for the technophile persona and a moderate persona weight the executive persona. This person might be very interested in technology news, and somewhat interested in financial news.

Using a combination of several crowd-sourcing platforms, including Amazon's Mechanical Turk, we administered a short survey to determine the breakdown of our user population and their attitudes towards sharing in formation in a mobile application context. To make the notion of a persona information concrete for the user, we asked them several question about their employment situation, gender, etc. to crudely approximate their persona. Note that this is *not* the same as doing precise data mining on the client, so, unsurprisingly, in some cases this inferred persona information was deemed imprecise. In later questions about persona sharing, this concrete persona information was mentioned explicitly. Moreover, survey questions were unfolded one-by-one, resulting in a slightly different workflow based on the answers, and preventing the users from revising their answers based on later questions. To obtain these values, we asked several qualifying questions such as whether the user had a smart phone and installed app, only counting those user's responses as valid for the purposes of determining their sharing preferences.

We summarize the results in Figure 3. Overall, the most apparent result is that 39–45% of survey takers are more willing to share persona data, whether in general or for the purposes of personalization, compared to the familiar baselines of location data and fine-grained location data. We feel that many users understood that their is more value and less sensitivity in sharing persona data compared to location.

3.3 Design Choices

Advantages of performing personalization within the mobile OS as opposed to the cloud, which is often the status quo, or within apps, are as follows:

- within the OS we greater visibility than within any application or a server in the cloud;
- sensitive data does not leave the device; instead, sensitive data is declassified by being distilled to personas, as explained below.

Where to personalize: A natural question is *where* to perform the personalization. While argue that mining be done on the device itself, we also we also strongly advocate performing personalization on the device, as shown in Section 5. However, we realize that an important practical alternative is server-side personalization. One advantage of doing things on the server include lower latencies: a personalized news reader such as The Early Edition[1] can pre-compile today's personalized newspaper for each persona and just deliver the right version when the user shares persona information.

Design alternatives: While our overall vision is quite general, the implementation described in this paper provides a fixed set of collectors and allows for actors that query personalization data through a fixed set of APIs, provided to applications. The decision to go with this more limited approach is justified by our desire to provide a simple and understandable interface to both users (Section 3.2) and developers (Section 4).

Our miners are part of a mobile OS *service*, allowing for easy data separation. We envision a more general system that would allow *installable* OS-level miners that would process, say, the user's email stream or derive more fine-grained information about the user's travel behavior from their Facebook check-ins. Of course, with this extra power comes the danger of sensitive user data being leaked by potentially untrusted collectors.

Extensibility: Much work has been done on verifiable extensions, in the contexts as diverse as OS drivers and browser extensions. In our case the problem is especially difficult because 1) we need to worry about *information flow* and not *access control* properties, and 2) the notion of adequate data declassification in general is very difficult to establish. Some proposed approaches involve static analysis [4], type systems [10], and runtime information flow tracking.

While all three suffer from problems of incomplete specification, with a single missing tainted data source leading to exploits [22], static analysis and type systems also are subject to precision and usability challenges. Runtime analysis can incur a non-trivial overhead. Unfortunately, none of these approaches seem ready for practical deployment for regular mobile app developers.

A more manageable task for future work is to alert the user when persona information *leaves the device*, in addition to alerting them when the persona information is first obtained; this can be accomplished with runtime taint tracking [7, 17].

3.4 Value Proposition

Our goal with MoRePriv is to provide an attractive value proposition for all major categories of constituents within the mobile ecosystem: users, developers, and third party data and ad providers. We summarize the benefits below.

Users: Users gain a clear way to communicate information about themselves for the purposes of personalization without revealing too much about themselves. Users can also *audit* persona information released to every application, to understand when each release has token place. Note that MoRePriv is designed to be combined with other privacy filters, to remove existing tracking, if present. To make it easier for users to understand, we model the way persona information is surfaced to the user after GPS location data; Figure 4(a) shows an example of a persona prompt within an app.

Developers: The main advantage for app developers is the MoRePriv API. It represents a single OS-level data source for personalization (as opposed to application-specific, ad-hoc information sources) allows seamless and uniform functionality for a single user across many applications, devices, and platforms. Easy for developers to use for personalization and skinning. Extension points [8] are possible.

The *cold start* problem common in many personalization tasks is largely addressed, because of an OS-wide user personalization context, which apps can easily take advantage of. Thus, applications can plug into an existing source of data about the user. Finally, OS-wide data collection for a single repository allows richer and more accurate profile information about the user to be collected than any single application can accomplish.

Third party and ad providers: The value proposition to data providers is that they are no longer responsible for storing "toxic" user data and cross-correlating it across a set of user interactions. Given the mounting pressure by legislators in the US and Europe to limit the impact of online tracking as well as mobile app tracking, this can we welcome news. Instead of trying to engage in user profiling, which is 1) costly, because it requires maintaining or buying data center capacity, developing custom software, and paying data mining and support personnel, and 2) dangerous, because multiple laws[2][3] ,[4] of different localities must be respected and there is

[1] http://www.glasshouseapps.com/the-early-edition/

[2] http://www.truste.com/blog/2012/03/02/mobile-app-privacy-policies-are-now-the-law

[3] http://www.applicationprivacy.org/?page_id=39

[4] http://www.infolawgroup.com/2011/05/articles/data-privacy-law-or-regulation/mobile-location-privacy-opinion-adopted-by-europes-wp29

always a danger of being limited later in time, a clear alternative is to use the information provided natively, by the mobile OS.

3.5 Data Protection and Privacy

So far, our discussion of MoRePriv has primarily focuses on personalization benefits of our approach. In the rest of this section, we focus on privacy-related advantages. In the past several years, a number of projects have examined data leakage potential on the mobile, including mobile information leakage [7, 9], leading to privacy-enhancing technologies designed to guard against such data leaks, usch as AppFence [15].

Privacy by design: We want to draw a contrast between that line of research and MoRePriv, which is envisioned as a privacy-by-design approach. The goal of MoRePriv is to *shepherd* the different constituents of the mobile ecosystem into compliance. MoRePriv can and should be combined with privacy monitoring and privacy leak detection tools, but we see that as an orthogonal concern.

Data stored at the OS level: Our implementation of MoRePriv maintains a vector of persona weights for each user at any given time. This vector is maintained serialized as part of the OS service that MoRePriv implements. To make this maintainence an incremental process, our implementation records two values for each persona p:

1. *relevance*: sum of interest scores from classifier C_p, s_p
2. *support*: number of elements that have been scored by C_p, called n_p.

Intuitively, s_p indicates a raw score of how closely the persona matches the user, while n_p indicates the amount of evidence to support that score. The persona weight is given by the fraction s_p/n_p. Thus, only two data vectors per persona are stored. Note that these vectors are never directly shared with user mode apps.

Use of personas limits information leaks: At the core of MoRePriv design is the use of personas, which provides a degree of pseudonymity [20]. Sensitive data is distilled by MoRePriv to a restricted, deliberately coarse-grained persona, and is never given to user mode apps directly.

A natural direction to consider is implementing an information flow restriction approach to mitigate leaks of persona information. While many research efforts have argued for information flow and tainting approaches, in both static and runtime flavors at levels varying from hardware to the application runtime, practicable adoption has been slow. We argue that this is because it is very difficult to these systems without causing numerous false positives or tolerating many false negatives.

In our model, apps are allowed to leak the most relevant persona of the user if they so desire. However, we consider the consequences of that to be relatively benign, especially when compared to the current practice of user monitoring built into many of today's apps (Section 6).

(a) Requesting access to persona information in an app. (b) Personas used in MoRePriv.

Figure 4. MoRePriv User Interface

Permissions: As Figure 4(a) illustrates, access to persona data in apps is guarded with a permission prompt, similar to that used for obtaining location data. We believe that this is part of responsible disclosure: the user is informed of persona data access and is given the opportunity to opt in.

Furthermore, at the level of application manifest, access to MoRePriv's persona data requires statically declared permissions. We envision extra scrutiny, code review, and testing being applied to apps that request persona data by app marketplace maintainers.

Data synchronization: While the default storage strategy is to keep the interest profiles local, on the current device, it is entirely possible to synchronize them — in an encrypted form — with the cloud. This is not unlike the approach used in Apple's iCloud for synchronizing application settings, etc. However, unlike application settings, persona information encroaches on user privacy considerably less.

In addition to synchronization across multiple devices, some desktop, some mobile, some tablets, cloud synchronization also serves as a backup. Precedents of this exist in several domains, including bookmark synchronization, Microsoft Office setting synchronization, Dropbox, automatic note synchronization with Windows Mobile phones and Windows Live, etc.

4. Implementation

In order to test the effectiveness of personal preference miners, we instrumented Windows Phone 7.5 (Mango) to capture several important *personalization signals*, sources of data that indicate likely user preferences. We then use these signals to locally classify the user with respect to the given personas. There are two facilities for personalizers in MoRePriv: a privileged service to perform automatic personalization within the OS, and a set of APIs that give third-party applications limited access to the user interest profile. Although we chose to implement MoRePriv on top of the Windows Phone, a similar implementation on Android or iOS is possible.

4.1 Personalization Signals

In order to assign relevance scores to each persona, MoRePriv needs data to classify. Here, we leverage our position at the OS level: all user information must pass through the operating system in order to be consumed or produced by the user. However, one must be careful of *which* data stream to collect: a poor choice can slow down the device or introduce noise. One must also consider *how* data is collected: the collection mechanism should be positioned at a level of abstraction such that the data has appropriate context. For example, one may be interested in mining the text of web pages that the user views, but if the miner interposing at a low level of the protocol stack, bytes of text will be indistinguishable from bytes of an image.

In our implementation, we capture five distinct personalization signals, as listed in Figure 5. We briefly discuss our approach to mining each of these signals on Windows Phone, and we mention how equivalent signals could be captured for an Android Device.

Facebook and Twitter: A unique feature of the Windows Phone is that several popular networking features are integrated directly into the operating system and organized into the *People Hub*. The intention of the People Hub is to organize social updates in a single, unified feed called the social feed, which is updated automatically. The social feed is a good target for mining because it is a rich source of structured user data. We implemented a miner for Facebook by reading social feed data from the Facebook service, consisting of "likes", posts that the user made, and posts that others made to the users wall.

There is no direct analogue to the People Hub on Android. However, since the account credentials are stored in the `android.accounts.AccountManager`, an Android device could make separate queries through the APIs exposed by high-value services like Facebook and Twitter, and classify the results of those queries. It should be noted that this approach loses an advantages of MoRePriv, namely that it does not consume any additional network bandwidth.

SMS: due to the simplicity and inherent lack of structure in SMS messages, we implemented our miner by interposing on the SMS handler in native code underlying the application framework. Alternatively, similar modifications can be made from within the C# core libraries to read SMS messages. A technique to implement a similar miner in Android would be to be to periodically query the `ContentResolver` for SMS content, and classify each SMS message in turn.

Email: Sending email is exposed to third party users via the `Microsoft.Phone.Tasks` classes such as `EmailComposeTask`. However, in order to capture simplify capture of both outgoing and incoming email, we instead interpose on the internal implementation of SMTP. Although we do not treat fields such as the subject and differently from text in the body, interposing on SMTP allows us to avoid classifying noise, e.g. attached images.

HTTP Traffic: Unlike SMS, HTTP has structure that cannot be ignored. Windows Phone framework, upon which apps are built. Fortunately, the Windows Phone passes information to an HTTP handler which parses the structure of the message. By interposing on the parser as it parses text, we can gather relevant web text without adding significant noise from non-textual HTTP traffic.

These signals demonstrate an advantage of performing signal capture at the Operating System level: since the OS and framework have a very high level of privilege, the user must already trust these components to handle personal data. As such, the signal capture mechanisms are already within the user's trusted computing base.

Furthermore, instrumentation at the OS level has the unique advantage of being able to integrate multiple data sources together. This is important for several reasons. Firstly, even very rich data sources can suffer from a cold-start problem, but are useful in aggregate. Figure 6 shows the technophile relevance scores for the example user with a strong interest in technology. The "wall" line represents the relevance score as posts on the user's Facebook wall are added. The "posts" line represents the relevance score as posts made my the user are added. The "likes" line represents the relevance score as the user "likes" additional things. In all three of these cases, it takes a fair amount of data for the signals to converge.

To drive this point home, we conducted a study to see how strongly a user with a strong interest in technology would be classified based on their interaction with various web sites or mobile apps. Figure 7 lists this user's technophile weigh (relevance computed by the appropriate Bayesian classifier) as they interact with

Miner/data source	Description
Email	Bodies of all incoming and outgoing email messages
SMS	Text of incoming and outgoing text messages
Facebook	Facebook "likes", posts by user, posts of others on user's wall
Twitter	Tweets posted to user's feed
Network interface	All other HTTP traffic to and from the device

Figure 5. Personalization miners in MoRePriv.

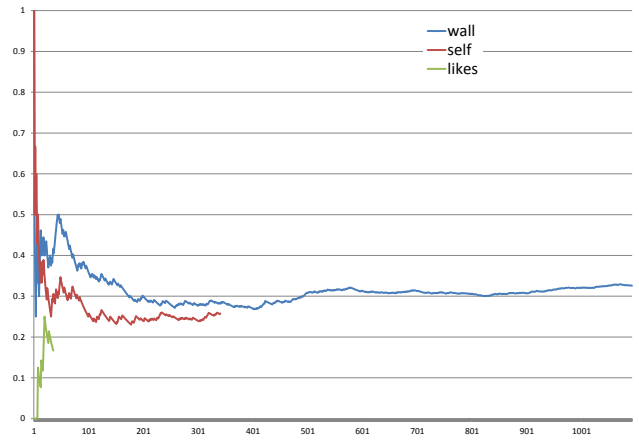


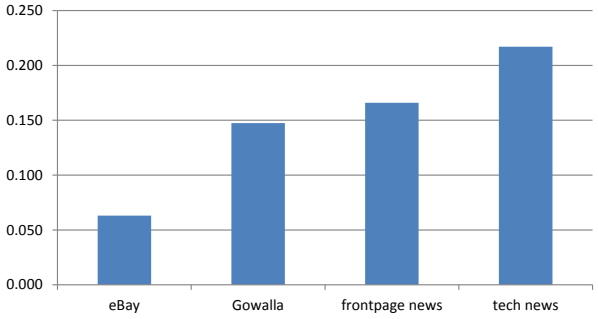
Figure 6. Facebook data convergence.

different sites and apps. While we would expect this score to be high, the quality of different signals taken in isolation varies significantly, as shown in Figure 7a. However, as shown in Figure 7b, *combining* these signals together can boost the correct relevance score even in the face of highly irrelevant signal data, such as data from eBay.

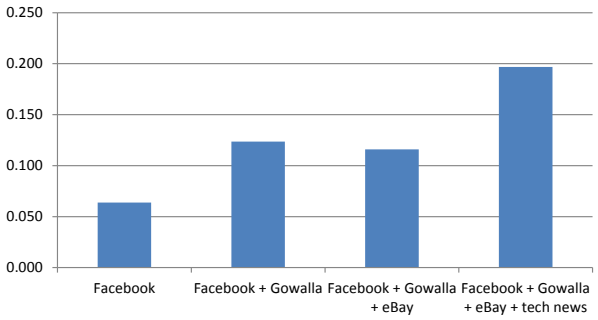
In MoRePriv, we give the user the option to switch data collection on and off (see Figure 9 for an example), but we envision a use case in which the data collection is always on, refining each persona's relevance score as the user interacts with their mobile device.

4.2 Building MoRePriv Classifiers

Each MoRePriv persona is represented by a Naive Bayesian classifier. We trained our classifiers offline on manually curated lists of words obtained for web pages relevant to each profile, e.g., `techcrunch.com` for technophile and `espn.com` for football dad. An alternative source of such pages are taxonomies such as the Open Directory Project (ODP).



(a) Sources compared.



(b) Sources combined.

Figure 7. Sources of personalization data.

This resulted in thousands of words per persona, which were subsequently used to obtain the probabilities $P(w_i|C_j)$ for each attribute word w_i and each persona C_j. This classification data was then loaded into an OS-level service.

Note that while building each classifier can in principle be a time-consuming task, especially if a large volume of training data is used, applying classification to a piece of text is very fast. For example, for find the most relevant profile for a piece of text, we tokenize it into words and perform a simple log-likelihood addition for each persona, maximizing over that value.

4.3 Universal Personalization

In the context of MoRePriv, we implemented *automatic* universal personalization within the OS. To accomplish this, we modified the Windows Phone C# framework upon which apps are built. We focused on reordering lists such that elements that are of the most interest to the user are at displayed at the top, while items of less interest to the user are be kept at the bottom. We modified internal widget classes such as `System.Windows.Controls.ListBox`, which is (directly or through the use of a subclass) used in many third party apps to display lists.

The standard `ListBox` kept an internal reference to an `ItemSource`, the list of elements to be displayed on the screen. Our modification added a second, *personalized* list to the `ListBox` that included all elements of the default list, but ranked by their relevance to the user profile. When the element is drawn to the display, the elements are drawn with respect to this second ordering, rather than the default ordering. An important consideration for universal personalization is not to personalize "too much". For example, if automatic personalization were to be applied to an alphabetized list, the alphabetic ordering would be lost. In light of this consideration, we attempt to detect if a list has been sorted, and if so we do not use the personalized list ordering to draw elements, instead relying on the ordering of the internal `ItemSource` list. We have made a small number of changes to the internal C# classes to inform our universal personalization mechanism that a list has been sorted, such as modifying the `List.Sort()` method to set a `sorted` flag on the list. Before performing the personalized sorting, classes like `ListBox` will first check to see if the `sorted` flag has been set.

4.4 OS-level Service

Positioning MoRePriv within the OS provides an opportunity to collect a great deal of data to build a user interest profile. However, it also provides an opportunity to perform personalization on user-level apps without any modification of the app itself.

To explore automatic personalization, we altered the Windows Phone C# framework to reorder lists in the application UI, as described in Section 4.3, based on the persona weights. For legacy applications such as news readers, this has the effect of not only reordering the order in which stories are displayed (stories more relevant to the user's interests are shuffled to the top), but also reordering entire categories of subjects such that the "technology news" category page of a news reader app appears earlier in the menu than the "arts section" for a technophile. Finally, in order to give users power over how the service is used, we allow users to toggle two independent facilities of MoRePriv:

- Users may switch personalization on and off. When personalization is off, the user will have the regular Windows Phone experience. When personalization is on, OS-level personalization is enabled and apps have access to the user's persona.
- Users may switch data collection on and off, thus allowing them to freeze their profile scores. This allows users to indicate to MoRePriv that it should not track any behavior of the user until persona refinement is re-enabled. This is a form of a privacy mode, similar to those supported in modern browsers.

4.5 MoRePriv APIs

MoRePriv also exposes APIs to third party developers that allow application-specific personalization. There are four API functions listed below. These APIs can be accessed via a user-mode library that can be bundled with an app. Consequently, a single app can be written to the MoRePriv APIs that will work on a standard image of the Windows Phone without the MoRePriv enhancements (albeit without personalization):

- `IsMoRePrivEnabled()` returns true if personalization is enabled. We allow users to toggle personalization on and off as part of the MoRePriv configuration UI.
- `Classify(String s, Object o)` classifies the relevance of `o` to persona `s`. For example, `Classify("technophile", "Computer")` will return a high value, because computers are of high interest to technophiles. Note that this call does not reveal any information about the user to the app, it is simply a convenience method to allow apps to classify objects.
- `TopProfile()` return the most relevant profile to the user if personalization is enabled, and `null` otherwise.
- `Ignore(Object o)` Informs MoRePriv not to apply OS-level personalization to `o`. This allows developers to bypass the GUI features such as automatic list reordering.

Example 1 Accessing the MoRePriv Profile
The example below shows an instance of application *skinning* using profile information that is obtained through a system call on lines 2 and 5. Depending on the current persona, a different background is used for the app. □

```
1 var bitmap;
2 if ("Technophile".Equals(MoRePriv.TopProfile()))
3     bitmap = technophileImage;
4 if ("Soccer Mom".Equals(MoRePriv.TopProfile()))
5     bitmap = soccerImage;
6 ...
7 var personaBG = new ImageBrush(ImageSource=bitmap);
8 app.RootFrame.Background = personalizedBG;
```

5. Personalization Scenarios & Studies

We discuss news personalization in Section 5.1, app skinning in Section 5.2, and highlight some other personalization opportunities in Section 5.3.

5.1 RSS News Feed Personalization

To test the usefulness of the MoRePriv APIs, we built a custom RSS reader called MoRSS. This app pulls stories from 10 RSS news feeds, and samples from these

feeds to display a list of stories to the user. MoRSS disables the OS-level GUI enhancements described in Section 4. Instead, MoRSS relies on the built-in table in Figure 8 to rate how interesting each of the RSS feeds that MoRSS subscribes to will be to a profile.

MoRSS can operate with no personalization, in which case stories from each RSS feed will be sampled uniformly and displayed to the user in the order in which they are sampled. When personalization is enabled, MoRSS queries the MoRePriv API to determine the top profile of the user, and then samples according to the column of Figure 8 for that persona.

Figure 9a shows an example screenshot of MoRSS with no personalization applied. In Figure 9b, the same set of stories are sampled according to the interests of the soccer mom persona column of Figure 8, which places an emphasis on Health and Entertainment stories. Similarly, Figure 9c shows the same set of stories sampled by the interests of the technophile column.

This demonstrates the advantages of exposing limited information to third party applications. Developers have the flexibility to reinterpret the top profile in any way that they see fit. Apps such as MoRSS are free to sample tech stories for the soccer mom, even though the built-in Bayesian classifier for that profile does not have tech keywords. Furthermore, the personalization can be done in a privacy-preserving way: MoRSS uses client-side personalization, so even the owner of the RSS feeds cannot learn the top profile of the user from the requests that MoRSS makes.

Example 2 Accessing MoRePriv Classifiers MoRePriv also simplifies the development of personalized apps by exposing a classification service to the developer, rather than forcing developers to include general-purpose classification algorithms in their apps. MoRePriv allows developers to query how relevant an entity is to each persona, and then use that value to determine if the entity will be of interest to the user. MoRSS takes advantage of this when the user adds a new RSS feed to their RSS stream, as shown below.

```
1 var top = MoRePriv.TopProfile();
2 var storyText = RSS_NewsItem.getText();
3 var weight = MoRePriv.Classify(top, storyText);
4 if (weight > THRESHOLD){
5  storiesToDisplay.append(RSS_NewsItem);
6 }
7 MoRePriv.Ignore(storiesToDisplay);
```

MoRSS gets the top profile for the user on line 1 and then determines how relevant a given story is to the top profile on line 3. If the story is considered to be highly relevant, as defined by some threshold value on line 4, it will be displayed. On line 7, MoRePriv is notified that the list of high-interest stories should not be subject to universal personalization, since it is constructed with custom, fine-grained personalization in mind. □

News topic	activist	bachelor	business exec	football dad	retiree	soccer mom	technophile	tween
Health	3	3	4	4	10	7	2	1
Tech	4	6	5	5	3	4	10	8
US	9	4	7	6	6	4	3	2
Business	4	5	10	5	5	2	6	1
World	7	2	4	1	2	2	2	1
Entertainment	0	7	3	4	4	6	5	5
Science	2	3	3	1	3	2	6	4
Society	6	3	2	2	3	3	2	3
Politics	10	4	6	5	5	3	2	1
Sports	0	7	4	10	4	2	5	7

Figure 8. News personalization parameters for Section 5.1.

5.2 Application Skinning

Though we have focused primarily on personalization as it related to networked device use, OS-level personalization has broad applicability. To demonstrate these possibilities, we implemented a simple calculator using the MoRePriv API. Figure 10a displays the default calculator with no personalization.

When personalization is enabled and the top profile is a tween, the calculator is whimsically re-skinned for a pre-teen girl. When the top profile is a retiree, the same calculator goes into a high contrast, high usability mode in which text size is increased, as shown in Figure 10b.

MoRePriv provides an alternative to providing complicated configuration menus to users who nonetheless prefer different configurations. Although the calculator will perform personalization at each run, an alternative would be to use user profile data to provide an initial configuration that is likely to be close to what the user wants, and allow her to tweak configuration options from there.

5.3 Other Personalization Examples

In the rest of this section, we focus on some prominent examples of personalization in the rapidly growing mobile space, and discuss their implications. This is not meant to be a comprehensive account, but rather a representative sample highlighting some of the more significant advances.

5.3.1 Word completion, spelling checking, and voice recognition

Given the small form factors of mobile devices, it is important to streamline common tasks, such as typing, for the user. For example, having typed `decla`, the completion might be `declaration` for a user interested

(a) No personalization. (b) Personalization (soccer mom). (c) Personalization (technophile).

Figure 9. RSS news reader.

(a) No Personalization (b) Personalization (tween) (c) Personalization (retiree)

Figure 10. Calculator personalization.

Permission	Ad-only
android.permission.INTERNET	699
android.permission.ACCESS_FINE_LOCATION	362
android.permission.VIBRATE	128
android.permission.READ_PHONE_STATE	425
android.permission.READ_LOGS	10
android.permission.ACCESS_NETWORK_STATE	483
android.permission.ACCESS_COARSE_LOCATION	502
android.permission.GET_TASKS	65
android.permission.ACCESS_WIFI_STATE	10

Figure 11. Reducing Android application permissions with ad library partitioning.

in *law* and `declamation` for a user interested in *public speaking* or *opera*.

Similarly, voice recognition software such as Siri on iOS or Dragon Dictate can be pre-populated with a different set of prior probabilities: a business executive is more likely to talk about a "cash management", whereas a technophile is more likely to talk about "cache management".

As the discussion above suggests, different personas are likely to use a different vocabulary, or at least use the same word with different probabilities. As such, spell checking can order suggestions differently for different personas, and perhaps even come with custom persona-specific dictionaries, augmenting the main one.

5.3.2 Suggested Web Sites and URLs

Today's smart phones come with browsers whose history and bookmarks are pre-polulated with a short list of sites that the phone provider thinks may be relevant for the user, such as `apple.com` for iOS. Based on the user's persona, the list of suggested sites for mobile browsing can be pre-populated differently. For a business executive, `ft.com`, `marketwatch.com`, and `forbes.com` are relevant, whereas for a technophile, `shashdot.org` and `techcrunch.com` will likely be of value.

The same principle applies to URL and search *suggestions* obtained by the mobile browser from the search engine such as Google or Yahoo. These suggestion lists can be re-prioritized based on the user persona.

6. Personalized Ad Delivery

Increasingly, mobile applications have embedded ads as a monetization strategy [14]. Much of the time, ad embedding is done by including a library that co-exists with the application. The library ecosystem is well-developed and crosses mobile platform boundaries, with most popular libraries such as AdMob providing versions that the developer can link with for iOS, Android, and Windows Phone. Other such libraries are provided by Flurry, mobclix, adwhirl, mobfox, and many other companies; the reader is referred to a survey by Grace *et al.* [9] for more details on the state of mobile advertising industry and ad libraries.

Existing challenges: Several problems both with confidentiality and integrity existing with the current approach have been identified, primarily stemming from library and app code not being properly isolated:

- Ad librares frequently access globally-identifiable data such as the device ID, known as `AndroidId` on Android or `DeviceUniqueId` on Windows Phone, or IMEI that is phone-specific. This allows cross-application mobile user profiling, a problem similar to tracking users on multiple sites on the web through the use of third-party trackers, but one made easier by the fact that correlation is trivial to establish.
- An ad library can force the application to increase its privileges. On the Android platform, permissions such as `INTERNET`, `ACCESS_FINE_LOCATION`, and `READ_PHONE_STATE`. This can lead to users deciding not to install the application for fear of what it may do to their device or with their data.
- Since the library is not isolated from the the core app, it may snoop on the rest of the application, exfiltrating sensitive user data. Indeed, it is fairly easy to develop a key logger masquerading as an ad library.

In many ways, the current situation with mobile ad libraries resembles that with third-party trackers that co-exist alongside first-party content on a web site. The disadvantages of unrestricted sharing of code and data between application logic and ad libraries are in reality quite similar. Both *integrity* violations such as the library interfering with normal app operation and *privacy* violations such as the ad library snooping on user data located within the app, are possible. Finally, *availability* challenges emerge if the ad library is hogging network resources, etc.

IFRAMES are a common browser-based solution that provides isolation, in that case to avoid data theft and arbitrary code injection [25]. In the rest of this section, we advocate a similar, albeit more special-purpose mechanism for isolating mobile ad libraries.

6.1 MoRePriv-based Design for Mobile Ads

An alternative design involves the mobile OS *explicitly* separating ad libraries from the rest of the app. The library may need more permissions that the application needs, including location data and persona information. However, the library will be prevented from getting global unique identifiers such as `DeviceUniqueId` and will contain no persistent state: the job of the OS will be to erase library-specific state. This will in turn lower app permissions and provide a degree of data isolation.

Advantages of MoRePriv: A key advantage of the ad separation approach is that the ad libraries can be jailed in several different ways. First, we can of course lower or deny permissions such as location permissions. However, in many cases that will lead those libraries to fail to cease to be useful. At the same time, we would like to protect the user from the aggressive tracking that some ad libraries perform. To do so, we must

- make the library compartment stateless;
- randomize user-specific identifiers;
- institute privacy-enhancing measures such as making location data more approximate [17].

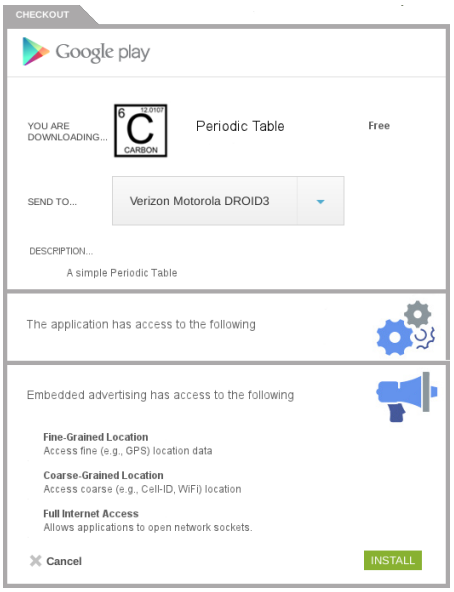
Monetization: A nice consequence of this design approach to the mobile platform is that both free and paid versions of the same app can be delivered as part of the same app package. This way, application updates to the free and the paid "pro" version do not get out of sync. Moreover, the app marketplace, with the developer's permission, can control the pricing model by watching the demand and competition, or using an auction, instead of the developer trying to do so crudely, based on their intuition and incomplete information about the market [5].

Moreover, this approach allows for a clear compromise between the app cost and the amount of advertising the user gets. For instance, the user may be presented with three versions of the same app, shown in Figure 12. However, in the MoRePriv approach, all of these can be generated from the same app package and the pricing can be selected by the user at installation time. For screenshots of sample marketplace interfaces automatically generated using this technique, please see our companion TR [2].

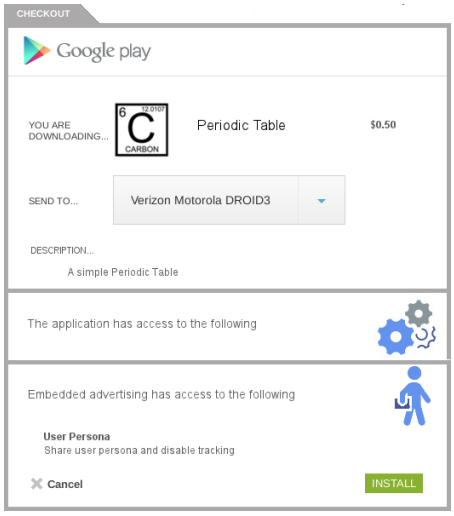
6.2 Experimental Evaluation

In order to determine how much opportunity for privacy and integrity abuse exists in the Android ecosystem, we developed a system to characterize how the permissions of an app are used with respect to advertising. Using the permission to function mapping of [27], we annotate functions of the Android framework with the corresponding permissions necessary to call that function. We will refer to functions that require some permission declared in the app's manifest as *permission sinks*.

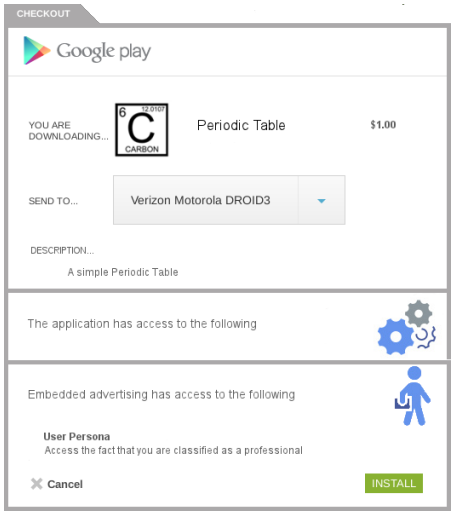
Our task is to determine the provenance of calls to sinks from the application. We do this by marking all methods in the application that are members of a known advertising library as *advertising sources*, and all other methods as *application sources*. We then perform a reachability analysis from advertising sources to permission sinks, and from application sources to permission sinks. If a permission sink can be reached from an application source, we consider the use of that per-



(a) Free version with advertising.



(b) Half-price version with persona information only.



(c) Paid version no ads.

Figure 12. Differently priced versions of the same app.

mission to have application provenance. If it can be be reached from an advertising source, it has advertising provenance. If the sink can be reached from both advertising and application sources, it has mixed provenance.

Experimental setup: Although there is a dearth of analysis tools for Android apps to perform our reachability analysis directly on the application's bytecode, there are an abundance of tools to perform such analysis on Java programs. Thus, we use the `dex2jar` decompiler to recover Java bytecode for our apps. We then construct our reachability graph and conduct our reachability analysis over the Java bytecode using IBM's WaLA tool [16]. In order to extract the manifest from the app, we use the `apktool` apk unpacker [1].

Experimental summary: We have experimented with 3,120 Android applications that we downloaded from the SlideMe third-party app store and were able to decompile and process using `dex2jar`. Out of these applications, 1,605 or 40% did *not* use any known ad libraries. Out of the remaining 1,876 applications, we found that 1,361 or 73% of the applications declared some permission that was used exclusively by bundled advertising libraries. By packaging these libraries separately, we could remove these permissions from the application's manifest completely.

Figure 11 shows a breakdown of these advertising-only permissions and lists the number of cases when they were used for the ad library only. By far the most common permission we were able to remove was `INTERNET`, in 699 out of 1,361 or 51% of the cases. This indicates that the only reason that the app needed network access was to fetch ads and report on ad clicks. Of course, removing this permissions has direct privacy benefits: the user no longer needs to be concerned about the app exfiltrating their data onto the network. Examples of application that benefit from this are puzzle apps such as One Piece Puzzle. Our analysis detected the presence of seven different ad libraries in this app, but did not find any non-advertising need for use of the network.

Representative examples: Another common permission was `READ_PHONE_STATE`, which protects access to the phone IMEI. This is one of the two most common ways to uniquely identify the user's device, the other being `AndroidId`, which does not require an access permission.

Because mobile ads are frequently location-sensitive, permissions `ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION` can often be removed as well, as the underlying application does not itself need location data. An example of this includes the Travel England app, which is a tourist's guide to places in England. The app integrates features such as posting on Facebook, and searching for information on specific tourist sites. However, somewhat surprisingly, the app does not use GPS functionality to locate the user. However, the embedded AdHub and Flurry advertising libraries do take advantage of these permissions in order to provide location-targeted advertising. This example illustrates that it is not enough to ascribe permissions to functionality based on expectations.

Interestingly, in 10 cases, we were able to remove the `READ_LOGS` permission, which allows apps to read low-level system events including debugging information. Overall, we have found 2,684 instances where a permission could be removed.

Treatment	Price	Advertiser separate	Ads preserved	Use persona
Default	Full price	no	yes	no
Separate	Full price	yes	yes	no
No ads	Full price	N/A	no	no
No ads + persona	Full price	N/A	no	yes
Free separate	0	yes	yes	no
Free together	0	no	yes	no
Half-price	1/2 price	N/A	no	yes
Half-price + ads	1/2 price	yes	yes	yes

Figure 13. Treatments used for user studies.

7. Related Work

Most efforts in this rapidly growing space focus on detection and prevention techniques for information leaks. Our work on MoRePriv takes a different position: there are legitimate reasons for apps to learn more about the user, and we want to provide extra value that goes beyond prevention, while preserving end-user privacy.

As a result, we see MoRePriv as complimentary to prevention mechanisms mentioned below. While privacy protection measure become more effective over time, we want to provide a legitimate path we can encourage app builders to use in order to perform personalization. We briefly cover mobile application privacy in Section 7.1 and on mobile ad privacy in Section 7.2.

7.1 Mobile Application Privacy

Monitoring: PiOS [6], which performs a reachability analysis over the control flow graph an an iPhone app to find privacy leaks. As we have also found in our analysis in Section 6, device identifiers are often leaked from the

device, though the situation is likely more pronounced than what we we see on Android because of `AndroidId`, which does not require a permission. TaintDroid [7] has a similar goal of detecting leaks, but it does so dynamically. While TaintDroid requires modifications of the Android operating system itself, our analysis is performed entirely offline.

Privacy-enhancing technologies: AppFence [15] allows fine-grained control over what permissions are used by the device. AppFence also allows for some control over the provenance of private data by restricting permissions such as limiting network communication to known advertisers, any third party. Unlike MoRePriv, AppFence does not offer an incentive for advertisers to agree to these controls, and it provides incentives for applications to engage in stealthy private data leaks, such as laundering data thrseparating advertising from regular content RePriv [8] explored personalization opportunities in the context of a web browser by building a *user interest profile* expressed using the Open Directory Project (ODP) taxonomy of interests, acquired based on the user's browsing history, by classifying the sites the user visits. Unlike RePriv, we focus on the mobile space. We go further by integrating our system directly into the operating system, drawing from more diverse data sources, and explicitly separating advertising from regular content.

7.2 Privacy in Advertising

One problem that has received much recent attention is that of delivering targeted advertisements to web users without violating their privacy, as summarized in a recent survey by Mayer *et al.* [23]. This is lead to the DoNotTrack initiative and a variety of policy and legislative pressures against advertisers and trackers.

An alternative direction perused by several researchers has advocated remedying the problem by storing the necessary sensitive personal data on the client, along with all ads to be matched [13, 18, 30]. When an ad is displayed, it is matched to personal information locally, thus sidestepping the need to leak to the ad network. Accounting and click-fraud prevention are addressed using either additional semi-trusted parties, or homomorphic encryption.

AdRisk [9] takes a similar approach to our own reachability analysis. However, the goals of the two projects are different. Whereas AdRisk seeks to characterize the permissions of common advertising libraries found in Android apps, it does so after decoupling the ad library from the app, and seeks to characterize the potential for abuse embodied by ad library has. Our reachability analysis seeks to compare, for each app, the aggregate effect of advertising permissions versus non-advertising permissions. AdRisk does not propose a mechanism such as MoRePriv in order to manage the relationship between advertising and application permissions beyond alerting the user to the presence of risky permissions in an advertising library.

Obliviad [3] proposes a cryptographically secure scheme using secure hardware on the server side to allow advertisers to serve targeted advertising to users. We differ from this work in that MoRePriv does not require the use of specialized hardware.

Developed contemporaneously with our effort on MoRePriv, AdDroid [26] and AdSplit [29] focus on the idea of separating ad libraries from the rest of the mobile app. While we share the spirit of this approach, as detailed in Section 6, we believe that we need to create a strong incentives for developers and ad library makers, and users. In MoRePriv, this is given in the form of persona data, which both provides much needed context for targeting, and allows the OS to both "police" the ad libraries and to create a natural monetary trade-off between app price and privacy guarantees.

8. Conclusions

This paper proposes operating system-level mechanisms that simplify building personalized applications. Our focus is primarily on mobile operating systems. We believe that at the level of the operating system these opportunities are largely untapped at the moment. This paper shows that personalization can be done quickly and effectively using personas, which provide a degree of pseudonymity and are easy for both users and the developers to understand.

We show how both OS-wide universal personalization and custom personalization can be done with little effort on the part of the developer, making us hope that persona and location information can become equally ubiquitous on mobile devices. When combined with a privacy filter for embedded ad libraries, we demonstrate how MoRePriv can be used to provide information needed for targeting while not violating user privacy, and reduce permissions in 73% of apps that use advertising. Finally, our user studies show that people are quite comfortable with releasing persona information.

References

[1] Android-apktool. `http://code.google.com/android-apktool/`.

[2] Anonymized for Submission. MoRePriv: Mobile OS Support for Application Personalization and Privacy (Tech Report). `http://www.arxiv.org/abs/1205.0467446v1`.

[3] M. Backes, A. Kate, M. Maffei, and K. Pecina. ObliviAd: Provably secure and practical online behavioral advertising. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2012.

[4] T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, and A. Ustuner. Thorough static analysis of device drivers. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 73–85, 2006.

[5] S. Dhar and U. Varshney. Challenges and business models for mobile location-based services and advertising. *Communications of the ACM*, 54(5):121–128, May 2011.

[6] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *Proceedings of the Annual Network & Distributed System Security Symposium (NDSS)*, Feb. 2011.

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, pages 1–6, 2010.

[8] M. Fredrikson and B. Livshits. RePriv: Re-envisioning in-browser privacy. In *IEEE Symposium on Security and Privacy*, May 2011.

[9] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Conference on Security and Privacy in Wireless and Mobile Networks*, Apr. 2012.

[10] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. Verified security for browser extensions. In *IEEE Symposium on Security and Privacy*, May 2011.

[11] S. Guha, B. Cheng, and P. Francis. Privad: practical privacy in online advertising. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 13–13, 2011.

[12] S. Guha, A. Reznichenko, K. Tang, H. Haddadi, and P. Francis. Serving Ads from localhost for Performance, Privacy, and Profit. In *Proceedings of the 8th Workshop on Hot Topics in Networks (HotNets)*, Oct 2009.

[13] S. Guha, A. Reznichenko, K. Tang, H. Haddadi, and P. Francis. Serving Ads from localhost for Performance, Privacy, and Profit. In *Proceedings of Hot Topics in Networking*, Nov. 2009.

[14] S. Han, J. Jung, and D. Wetherall. A study of third-party tracking by mobile apps in the wild. Technical report, University of Washington, Mar. 2012.

[15] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These arenŠt the droids youŠre looking for: Retrofitting android to protect data from imperious applications. In *Proceedings of the International Symposium on Information, Computer, and Communications Security*, 2011.

[16] IBM T.J. Watson. Watson libraries for analysis. `http://wala.sourceforge.net`.

[17] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster, and T. Millstein. Dr. Android and Mr. Hide: Fine-grained security policies on unmodified Android. Technical Report CS-TR-5006, University of Maryland, Dec. 2011.

[18] A. Juels. Targeted advertising ... and privacy too. In *Proceedings of the Conference on Topics in Cryptology*, Apr. 2001.

[19] A. Kobsa. Privacy-enhanced personalization. *Communications of the ACM*, 50(8):24–33, Aug. 2007.

[20] A. Kobsa and J. Schreck. Privacy through pseudonymity in user-adaptive systems. *ACM Transactions Internet Technologies*, 3(2):149–183, May 2003.

[21] H. Lam and P. Baudisch. Summary thumbnails: readable overviews for small screen web browsers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 681–690, 2005.

[22] B. Livshits, A. V. Nori, S. K. Rajamani, and A. Banerjee. Merlin: Specification inference for explicit information flow problems. In *Proceedings of the Conference on Programming Language Design and Implementation*, June 2009.

[23] J. Mayer and J. C. Mitchell. Third-party Web tracking: Policy and technology. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2012.

[24] J. R. Mayer and J. C. Mitchell. Third-party Web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy*, May 2012.

[25] J. R. Mayer and J. C. Mitchell. What we know: Third-party web tracking policy and technology. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[26] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: Privilege separation for applications and advertisers in Android. In *Proceedings of AsiaCCS*, May 2012.

[27] A. Porter Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *ACM Conference on Computer and Communications Security*, pages 627–638, 2011.

[28] E. Rich. User modeling via stereotypes. *Cognitive Science*, 3:329–354, 1979.

[29] S. Shekhar, M. Dietz, and D. S. Wallach. Adsplit: Separating smartphone advertising from applications. *CoRR*, abs/1202.4030, 2012.

[30] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings of the Network and Distributed System Security Symposium*, Feb. 2010.