Objective-Driven Context Mocking for Smartphone Users

Paper # 8

Abstract

Smartphones represent the most serious threat to user privacy of any widely-deployed computing technology. Today millions of people voluntarily carry an always-on, observant, powerful, and connected device with them at all times. This provides onboard smartphone apps with an ideal vantage point from which to invade users' personal lives. Unfortunately, existing permission models provide smartphone users with limited protection, in part due to the difficulty to users in distinguishing between legitimate and illegitimate use their data. A mapping app may upload the same location information it uses to download maps (legitimate) to a marketing agency interested in delivering location-based ads (illegitimate), while a pedometer processes the same accelerometer data it uses to count steps (legitimate) to determine the user's weight (illegitimate). As a result, smartphone users find themselves forced to make burdensome and error-prone tradeoffs between app functionality and privacy.

We propose a new approach, called DataDecoy. Instead of trying to limit apps' access to user data, DataDecoy turns apps' interest in data collection against them. By allowing substitution of real data streams with artificial or *mocked* data, DataDecoy allows users to manipulate impressions of their behavior in well-defined ways: to appear more fit, more social, or more on-time than they actually are. Instead of focusing on privacy, we explore providing users with an amount of management over their smartphone-derived digital identities that echos the control they already possess when using online tools such as social networks. We discuss the design of DataDecoy, which uses user-initiated context trace recording and replay to enable objective-driven context mocking. Our evaluation shows that users want to use DataDecoy, that DataDecoy can mock popular smartphone apps, and that DataDecoy is usable.

1 Introduction

As smartphone apps get smarter, they will be able to determine more about our lives through completely passive observation. Mapping services already know where we are, sensors reveal whether we walked or drove there, social networking apps examine our relationship with the person we came to meet, and smartphone-based payment tools [3] reveal what we did. As analytical approaches that fuse data from multiple sensors improve, smartphones will likely reveal even deeper facts about us: the strength of our friendships, the health of our lifestyle, our level of devotion to our job, even our level of happiness.

With the digital portraits painted by smartphones becoming ever clearer, we believe it is time to give users more control over their smartphone-derived digital identities. Many other components of our digital lives already provide ways to curate the information we provide in order to mold our digital personas. Users of dating sites post pictures that they hope others will find attractive, users of photo-sharing sites post more photos of themselves out performing interesting activities than of mundane sedentary ones, and users of social networking sites carefully choose the movies and music they list in their profiles to create a particular desired impression. While using these online services requires surrendering some amount of privacy, the active participation they require provides users with some control over what information they reveal and the impression they create. In contrast, the passive data collection that smartphones facilitate represents a dangerous simultaneous loss of *both* privacy and control.

Today's smartphone platforms have attempted to address this loss of privacy through permission mechanisms that limit apps' access to user information. Unfortunately, this approach has many well-documented problems: apps request permissions they don't need [11, 12], and users do not understand the implications of permissions that apps request [13]. The "take-it-or-leave-it" model used by Android provides no options for users uncomfortable with the permissions an app requests other than not to install it. Even a more selective "take-it-or-break-it" approach [17] that could allow users selectively to enable individual permissions is no cure. Users are still forced to make poorly informed tradeoffs between privacy and functionality, as it is unclear how an app might behave if denied access to information or fed random ersatz values. The result in practice is that users tend to give apps the permissions they request.

One fundamental problem with all permission-based approaches to protecting privacy on smartphones is that many apps legitimately require access to certain kinds of sensitive information to function properly. When I am lost, my mapping app must know where I am in order to route me to my destination. When I am monitoring my fitness, my pedometer must be able to access the accelerometer to count the number of steps I take each day. It is unreasonable to expect users to be able to distinguish between legitimate and illegitimate information requests, and burdensome and error-prone to ask them to enable data sources only when they feel comfortable with what a particular app is doing.

Our solution takes a different approach. Instead of focusing on privacy by limiting data collection, we aim to improve control by generating synthetic or "mocked" data to manipulate data-driven analytics as directed by the user, an approach we call *objective-driven context mocking*. In contrast to privacy, which aims to limit access to data, mocking reduces the power of legitimate data by injecting enough mocked data to achieve user-defined objectives. Unlike privacy, which requires hiding data and thus potentially impacting apps' functionality, mocking ensures that apps continue to function normally during each mocking session, making it simpler for users to understand and use.

Our paper makes the following contributions:

1. We introduce objective-driven context mocking, a new approach to protecting smartphone users' personal data that is orthogonal to privacy, and use several examples to illustrate the power of our approach.
2. We describe the design of DataDecoy, a system enabling objective-driven context mocking – that is, a system driven by users' objectives in crafting their digital identities. DataDecoy mocks context by capturing and replaying all information that could allow apps to pierce the mocking environment: location, network characteristics, and sensor data. DataDecoy's implementation consists of both Android platform modifications that allows mocked data to be fed to unsuspecting apps and an app that controls the mocking process by allowing users to record and replay mocking traces.
3. We evaluate DataDecoy and show it to be both desirable and effective. A survey of smartphone users provides evidence that many would like to alter how aspects of their digital identities appear to smartphone apps. Field testing of a DataDecoy prototype demonstrates that it can successfully mock several popular Android apps.

The rest of our paper is structured as follows. After motivating DataDecoy with several examples in Section 2, we describe DataDecoy's design and implementation in Sections 3 and 4. Section 5 evaluates our DataDecoy prototype, showing both that DataDecoy works and that smartphone users are interested in using the capabilities it provides. Section 6 raises and discusses some ethical and practical concerns raised by objective-driven context mocking. We review related work in Section 7, discuss future plans in Section 8, and conclude in Section 9.

2 Motivation

Recent studies show that privacy is one of today's smartphone users' top concerns with their devices, second only to battery life [4]. A plurality of 43% of users are *not* willing to share any information about themselves with a company in exchange for a free or subsidized app, despite ad-driven free apps being common on mobile app marketplaces. Smartphone privacy receives significant attention from both researchers and developers. Ironically, we believe that while privacy protection is a worthwhile goal, privacy itself may be part of the smartphone privacy problem.

The reason is that understanding privacy implications requires smartphone users to answer difficult questions. If I install and use this app, what will it be able to determine about me? How much of the data that this app is collecting is really necessary? What are the privacy implications of even the legitimate data that this app is collecting? Accurate answers to these questions remain elusive at best. There are billions of dollars at stake for companies in determining how to do more accurate mobile data analytics, and few if any have a business interest in divulging either how their algorithms work or what they really know about us. While new tools help smartphone users determine how much data smartphone apps collect, what is done with the data that is collected remains opaque. Users remain guessing and worried.

Obviously users can always choose not to use apps with which they feel uncomfortable. There are many projects looking at how to make safer app marketplaces by preventing malicious apps that *only* want to steal personal information. However, this focus on malicious apps obscures a harder truth, viz., even the legitimate data collected by non-malicious apps creates a privacy risk for smartphone users. For a typical user who wants to read email, browse the web, take pictures, and use social networking and messaging clients, legitimate data collection by legitimate apps still constitutes a significant risk to their privacy. Even after preventing unnecessary data collection by legitimate apps, a tradeoff between privacy and usability remains. The only options left to smartphone users at this point are to remove useful apps or to cease using their smartphone entirely – both unattractive.

It is at that point that mocking has a role to play. In contrast to the uncertainties caused by privacy, mocking provides control. Instead of wondering what information an employer required app collects on a bring-your-own-device, users can set up mocking objectives that ensure that it appears that they work regular hours. Instead of wondering which apps might be collecting information about their drinking habits, users can set up mocking objectives to conceal their visits to bars. We believe that this type of control over their digital identities will be appealing to users, since it is similar to the control they already have when interacting with other online services. While Facebook users know that Facebook is collecting data about them, they also exercise control over the impression they create. DataDecoy provides smartphone users with the same control over their smartphone-derived digital identities.

2.1 Mocking Scenarios and Types

Consider these four scenarios:

- Bob wants to appear more active. On Monday he takes a walk to get some exercise. The next day, he doesn't take a real walk, but while he is sitting at his desk his smartphone mocks another walk.
- Alice wants to appear more healthy, but on Monday she visits a fast food restaurant where she enjoys an unwholesome meal. As she eats, however, her smartphone mocks a visit to a nearby organic salad delicatessen.
- Teenager Jerry's parents use a smartphone app to monitor his late-night ventures and to ensure that he returns home by an imposed curfew. One night Jerry remains out on the town later-than-allowed with his friends, but his phone has already mocked him dutifully returning home on time.
- Carol's employer uses her phone to monitor her attendance. During the workday, she surreptitiously slips out for a latte with a friend. Meanwhile, her phone records her apparent continued presence at her desk.

These examples highlight the difference between *privacy* and *mocking*. None of our characters' objectives can be accomplished through privacy, since in each case achieving the objective requires using data to manipulate an app. Of course, Jerry could remove the app that his parents installed, as could Carol, but his parents and her employer would likely notice. In the cases of Alice and Bob, we can assume that they have been asked or incentivized to install these health-monitoring apps but may not feel fully-comfortable with their operation. Similarly, however, removing or disabling the apps may not be an attractive option. We include a more in-depth discussion of the practical and ethical implications of DataDecoy in Section 6 after establishing the feasibility and desirability of our system.

These scenarios also illustrate examples of two different types of mocking: record-and-replay and time shifting.

2.1.1 Record and Replay Mocking

In record-and-replay mocking the objective can be directly embedded in the mocked activity. Bob's recorded walk inherently makes him appear more active the more he replays it, and Alice's visit to the healthy restaurant makes her appear more healthy. In other cases, the objective can also be to obscure another activity, replacing something undesirable with something desirable. As Alice replays the desirable visit to the healthy restaurant, it provides cover for her undesirable visit to the unhealthy restaurant.

In other cases, DataDecoy users may simply want to have their smartphones mock staying in one place while they in fact go to another, as in Carol's example when she visits with her friend instead of working. While this behavior is similar to that offered by traditional record-and-replay systems, this variant requires DataDecoy be able to extend or loop a recorded session for an indefinite amount of time in order to mask an activity and to ensure time continuity between the mocking trace and the user's real activity.

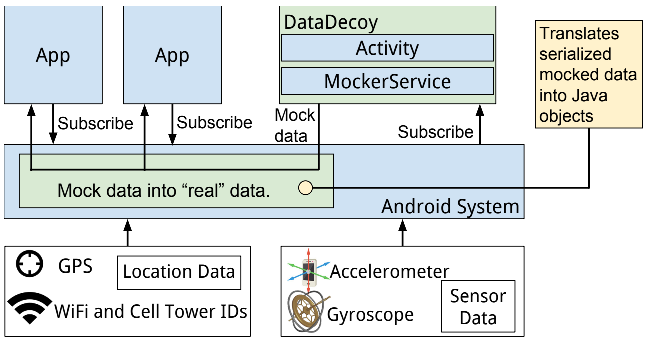


Figure 1: **DataDecoy Design.** Details are specific to Android but would be similar on other platforms.

2.1.2 Time Shift Mocking

In time-shift mocking, a user wants it to appear that something that happened at one time actually happened at another. Time shifting can move an activity later or earlier, the later exemplified by Jerry's example where he wants his parents to think that he arrived home punctually. We call the former *forward* time shifting and the latter *backward*.

While conceptually similar, forward and backward time shifting create different design requirements. Backward time shifting—making something that happens in the future appear to happen earlier—requires either being able to synthesize a transition from the current context to the mocked context or having a pre-recorded trace that accomplishes the same thing. In Jerry's example, in order to look like he returned home early, his smartphone must either be able to create a location transition from his current location to home, or he must have previously recorded this transition.

Forward time shifting is somewhat easier, since the user's real transition can be recorded, saved, and then held until the user is ready to replay it at a later point in time. If Carol wants to leave work early, she can record her transition to home but then delay it for several hours until the working day is over. Like record-and-replay, however, this type of mocking also requires DataDecoy to be able to dwell in a particular context while the user moves to another by looping a portion of a pre-recorded context.

3 DataDecoy Design

This section describes the design of the DataDecoy objective-driven context mocking system. We begin by developing a set of design requirements based on the mocking scenarios and taxonomy presented previously, and also outline the challenges of effective context mocking. We then describe how DataDecoy uses changes to the smartphone platform and a dedicated app to perform context mocking.

3.1 Overview

We offer the example of Bob from our earlier scenarios as an overview of how DataDecoy's components work together to deliver objective-driven context mocking. DataDecoy consists of two parts: modifications to the smartphone platform needed to record and replay mocking traces, and an app that interacts with the user to record mocking traces and control the mocking process. Figure 1 illustrates the interaction between the two parts of DataDecoy.

Once Bob has formulated his objective—to appear more active to his smartphone—he is ready to use DataDecoy. The first step is to determine what activity will accomplish this objective and to record it. In this case, Bob determines that frequent brisk walks will make him appear more active. Using the DataDecoy app, he initiates the context recording process and goes out for a model walk. DataDecoy records every piece of information it might need later during the mocking process: his GPS location, the cell towers to which he is connected, visible Wifi access points, and readings from onboard sensors such as the accelerometer and thermometer—all appropriately timestamped. Because DataDecoy does not know what apps will be running during future mocking sessions or what data they will request, it records as much information as possible even if it is not requested by apps during the original recording session. This data set comprises the *mocking trace*.

Once the trace is recorded, Bob can replay it as often as he likes. During the mocking process, DataDecoy exploits changes to the underlying smartphone platform to satisfy app requests for real data with time-shifted false numbers from the mocking trace. While the mocking session is active, the DataDecoy app displays a notification indicating that mocking is in progress and how much time remains before it finishes. After completion, DataDecoy stops returning mocked data and resumes returning real data to apps.

Based on this use case, we can enumerate several requirements for an objective-driven context mocking system. First, the user's objective must be determined and a mocking activity suggested. Second, a mocking trace must be recorded and linked to the user's objective. Finally, the trace must be deployed as needed to achieve the user's mocking objective. We describe how DataDecoy accomplishes these tasks below. We also address how DataDecoy overcomes two challenges that might allow apps to detect the mocking process: differences between the mocking trace and the real current context, and spatial discontinuities. Time-shift mocking has somewhat different requirements, and we discuss them separately at the end of this section.

3.1.1 Linking mocking traces and objectives

To begin, DataDecoy must be able to link mocking traces with user-defined objectives, so that it knows what trace will achieve each objective. In Bob's example, DataDecoy must be able to associate the trace of Bob taking a walk with Bob's desire to appear more active. This process has both a qualitative and quantitative component. Qualitatively, DataDecoy may suggest activities that would naturally be linked with a specific objective. If Bob wants to be more active, he should take a walk. If Alice wants to appear more healthy, she should eat at a healthy restaurant—and not eat at the fast-food restaurant. DataDecoy's app provides a library of objectives ("appear more fit") and associated suggestions for mocking traces ("take a walk"). We also expect users will be well-served by their own intuition.

3.1.2 Collecting, storing, and replaying traces

Second, DataDecoy must be able to collect, store, and replay mocking traces. Trace collection is currently initiated by the DataDecoy app and performed entirely at the app level. To ensure that during mocking DataDecoy can return data from any source consistent with the mocked context, DataDecoy currently enables all sensors that could provide relevant information and samples them aggressively, storing timestamped data in a set of local databases. This produces a high energy overhead for mocking trace collection, which we measure in more detail in Section 5.2.4, but ensures high mocking fidelity. As one optimization to the process of recording mocking traces, DataDecoy uses changes to the users current location to trigger the immediate collection of location-specific data. For example, when DataDecoy detects a location change it immediately stores new Wifi scan results and cellular signal strength information. As long as the user is not moving, these kinds of data are sampled more slowly. Because traces are recorded only once and replayed multiple times, and because the energy overhead of replaying traces is minimal, we consider the high overhead for trace recording acceptable.

Unlike trace collection, mocking trace replay requires platform support. DataDecoy modifies the underlying smartphone platform to add an interface allowing it to inject mocked data. Once the user begins replaying the trace, DataDecoy reads data from all sensor databases associated with the trace and uses this new interface to inject it into the platform. Any app requests for data contained in the mocking then return mocked data.

3.1.3 Initiating mocking sessions

Third, DataDecoy's app helps the user remember to initiate mocking sessions. Each recorded trace can be annotated with a frequency which DataDecoy uses to help prompt the user to deploy the trace. For example, Bob may want to go for a walk once every day, and by annotating the trace with his goal DataDecoy knows when to provide reminders.

3.2 Consistency Challenges

A significant challenge when mocking is addressing differences between the mocking context and the real context to ensure that mocking proceeds consistently. At present DataDecoy does not attempt to fully defend the mocking context from suspicious apps—we leave that challenge as future work. However, DataDecoy still attempts to ensure that the mocking context is consistent and does not create obvious problems or physical impossibilities that could either break app functionality or send an unmistakable signal that something unusual is happening. First, we look at how DataDecoy masks differences between the mocking context and the real context. Second, we address spatial continuity, a specific consistency problem facing the DataDecoy system.

3.2.1 Differences with the mocking context

Here we examine specific differences between the mocking context and the real context and address how DataDecoy deals with each case:

- **Location:** the phone is one place in the mocking context and another location in the real context. To ensure location consistency, DataDecoy collects all data associated with the mocked location. During the mocking session an app will not only have the mocked location coordinates returned, but will also see the same Wifi access points and be connected to the same cell tower with the same signal strength as it would at the mocked location.

- **Device configuration:** the accelerometer was not used during the mocking context but is enabled by an app in the real context. Here DataDecoy exploits the fact that it records all information about the smartphone while recording the mocking trace, meaning that it can handle requests to use any device feature during replay.
- **Connectivity:** the phone was connected during the mocking context but there is a different or no network connection available in the real context. There are two cases to consider here. If the smartphone has any connection in the real context, DataDecoy will allow apps to use that connection but return mocked connection *attributes*. So if the connection is actually over a Wifi network but only a 3G mobile data network is available, DataDecoy will establish connections over the available network but tell apps that they are connected over the mocked Wifi network. At present DataDecoy makes no attempt to alter connection properties such as latency or bandwidth of the real connection to match the mocked connection, and in some cases this is not possible. We leave dealing with attempts by suspicious apps to use these properties to pierce the mocking context as future work.

  A more difficult problem is providing a mocked connection when no real connection exists. Since this is impossible, DataDecoy simply synchronizes this aspect of the mocking context with the real context and does not provide any mocked networking information while a real network is unavailable. While this creates variance in the mocking context, networks naturally go up and down and a user may have disabled the phone networking manually, meaning that lack of networking connectivity at a location where it is typically available cannot necessarily be interpreted by an app to mean that mocking is happening. Smartphone apps have to naturally handle this kind of variation in networking conditions due to mobility and changes in user configuration.

It is also important to point out things that DataDecoy *does not mock*, including user interaction, battery level, and the microphone readings. While mocking user interaction may be necessary to mislead certain types of apps, it is not necessary to mock the apps that DataDecoy currently targets that collect and interpret data collected passively. More importantly, replaying interaction would prevent the user from using their smartphone while mocking was active.

Replaying microphone readings represent a similar challenge since the microphone is used by the phone feature of the smartphone which we expect that users will not want to disable during mocking. Unfortunately, not replaying certain aspects of the audio environment may allow other apps to detect a different between the mocked context and the real context, such as a user claiming to be at a loud bar but actually still at their quiet home. We are planning to address this current limitation of DataDecoy in the future, either by providing a pass-through for real context audio to apps where it represents input, or by merging audio recorded during replay with the audio from the mocking trace.

Finally, if replayed battery levels would represent another continuity challenge similar to the spatial continuity challenge presented next. If DataDecoy produced mocked battery readings during trace replay that represented the actual rate of battery usage during the trace, the system would be left with a discontinuity when the trace completed as the battery level would either have to abruptly increase or decrease to merge with the actual battery level. On the other hand, continuing to report actual battery levels during trace poses no threat to the mocking context and is actually more realistic, since the battery level will continue to track app usage of the device. For example, if the mocking trace was performed while no apps were rapidly using energy, and then apps did perform activities that led to heavy energy usage during replay, replaying the original battery drain would allow an observant app to identify a difference between the mocking and real contexts. Continuing to report actual battery levels is both easier and more correct.

3.2.2 Ensuring spatial continuity

Given that smartphones can and do track their users location, and that this information reveals a great deal about their lives, DataDecoy is designed to allow mocking location and user movement. However, this creates a continuity challenge when the mocking trace ends at a different place from the user's current location. We discuss in Section 8 how future version of DataDecoy will use user-generated mocking libraries to be able to synthesize mocking traces linking any two points where the user has previously been, but our current prototype has no good way to address this problem. And while we are currently focusing on mocking unsuspecting apps and not addressing all attacks apps could perform on the mocking context, sudden changes in location are both an all-too-obvious indication of mocking and might also cause some apps to malfunction.

Currently, DataDecoy works around this challenge by interacting with the user. While a mocking trace is being replayed, a notification is displayed informing the user of the time left before the mocking process completes and the distance from the user to the location where the trace ends. Once the trace ends, if the user's location is close to where the trace completes DataDecoy will simply allow the trace to end normally and merge the real and mocking context. If the user is not close to where the trace completes, DataDecoy generates an notification asking the user to either reach the correct location or allow the spatial discontinuity. Until the user responds to the dialog or reaches the required location, DataDecoy continues to mock them at the mocking traces final location, using the lingering capability described next. This also allows DataDecoy to perform backward time shifting on an existing trace. In Jerry's example, when it is time to return home he initiates a pre-recorded trace of his return. Once his trace reaches home, it will linger there until he arrives at some later time.

3.3 Lingering

In addition to the record-and-replay functionality we have already described, DataDecoy also supports *lingering*. Lingering is a mocking primitive that can be used in several ways: to time-extend a mocking trace, to conceal an undesirable activity, or to perform time-shift mocking. In the scenarios described earlier both Alice, Jerry, and Carol's mocking activities require this capability. Carol uses lingering to conceal her coffee break, Jerry uses it to time-shift his return home, and Alice uses it to time-extend her visit to the healthy restaurant to match the time she spends eating fast food.

To linger, DataDecoy records a small amount of context at a particular location and then replays it repeatedly. To make the data delivered to apps during the lingering process more realistic, and prevent apps from detecting the mocking process by observing repeated readings, DataDecoy injects noise into the data returned during the lingering session, performing small changes to the reported location, sensor readings, Wifi scan results, and network signal strengths.

To time-extend a trace, the app allows users to indicate linger points during trace recording. During replay, once the trace reaches a linger point DataDecoy will linger until instructed to proceed by the user. Once Alice is ready to leave the fast-food restaurant, she tells DataDecoy to proceed past the linger point she inserted into her trace of visiting the healthy restaurant. To conceal an undesirable activity, the DataDecoy app allows lingering to be initiated at any time. A small amount of context is recorded and then replayed until the lingering session is canceled. So Carol can initiate the lingering session at work, and then bring her phone with her to her coffee break while appearing to remain at work.

Forward-shifting a trace is a three-step process. First, DataDecoy collects a small amount of context at the current location in order to linger and begins the lingering process. Second, DataDecoy records a user moving to a new location while continuing to return lingering data. Finally, once the user is ready to merge their real and mocked context, DataDecoy stops lingering and begins replaying the transition trace until the user reaches their current location.

4 Implementation

We implemented a DataDecoy prototype on Android 4.2.2 "Jelly Bean", Android being the only open-source smartphone platform permitting the platform modifications DataDecoy requires. Our current prototype supports recording and replaying context traces and mocking location, available networks and signal strengths, and sensors including the GPS, accelerometer and gyroscopes. We have deployed our prototype on the Samsung Galaxy Nexus smartphone [7] which was used for the experiments in Section 5.

We considered implementing DataDecoy support in two ways: either by modifying Android platform services, or by making changes to the underlying Linux kernel. Most Android platform services provide thin wrappers around low-level Linux interfaces in order to provide and protect Android interfaces to modifying core smartphone features. For example, the `WifiManager` Android interface for switching access points translates requests from apps with permission to use the interface into the appropriate manipulations of the wireless connection state using tools that unprivileged Android apps lack the permissions to use. Because Android apps typically use Android's service interfaces to collect information about the device, such as determining the access point that the smartphone is currently associated with, it is possible to implement successful mocking with these services and fool many apps.

Unfortunately, the underlying Linux interfaces on Android leak a great deal of information about the state of the system that apps could use to pierce the mocking context. For example, reading `/prot/net/arp` allows an unprivileged app to determine the access point the smartphone is associated with in the same way as a call to the `WifiManager` Android service. So implementing mocking *only* within the Android platform is not sufficient to fully secure the mocking context, since apps may be able to bypass services participating in the mocked process. The most secure way to implement mocking would be to make changes to the Linux kernel itself to ensure that all information provided by the system would be consistent.

At present, however, our current DataDecoy prototype is implemented as a set of changes to Android. This is for two reasons. First, DataDecoy is currently designed to fool unsuspecting apps, and we have left as future work the task of exploring ways apps could attack the mocking context and effective DataDecoy countermeasures, including moving mocking support into Linux itself. Our evaluation demonstrates that platform changes are sufficient to mock many different apps. Second, modifying Android reduced the developer effort needed to produce a working prototype allowing us to test and better understand the system.

The architecture of DataDecoy consists of two phases, one sitting in the app layer and the other in the Android platform. We handle user interaction, data logging, and data replaying in the app layer. In the platform, we have made modifications allowing DataDecoy to notify user-installed apps of location and sensory updates through their respective callbacks.

Users interact with the app itself, which allows for initiating the recording and replaying processes, and managing objectives. A user must create an objective before recording data; otherwise the system cannot associate a trace with a particular goal. During a recording session, we aggressively log device events to the application-layer database. DataDecoy registers a `LocationListener` with the `LocationManager` that records `onLocationChanged` at a rate of 1 `Location` per second—our listener also logs `onProviderDisabled`, `onProviderEnabled`, and `onStatusChanged` events. For each `onLocationChanged`, we also scan for nearby wireless acces points through the `WifiManager` and collect the `CellLocation` provided by the `TelephonyManager`. We also register a `SensorEventListener` with the `SensorManager` to collect sensor events—such as `onAccuracyChanged` and `onSensorChanged`—at a defined rate of `SENSOR_DELAY_GAME`, which is somewhere between 37–39 milliseconds. For all logged data, we serialize it into the database such that it can be reconstructed as a Java object later. In the background is the replay service, an Android `Service`, that manages platform clients and their respective publish-subscribe channels for replaying sensory data. When replaying data to the platform, we update relative data, such as timestamps, to reflect current device time.

		Address		Social Network		Activity Level		Income		Weight	
Accuracy		exact	81	five best friends	49	**quantity and type**	20	**$1**	6	**1 lb**	8
		street	6	some friends	25	how much	20	**$100**	24	**10 lb**	12
		neighborhood	12	how social	23	activity level	43	$10,000	31	category	39
		nothing	0	nothing	2	nothing	14	nothing	37	nothing	39
Comfort	High		16		19		25		9		36
			7		12		23		13		14
			14		29		28		23		19
			24		19		9		10		9
	Low		37		18		13		42		19
Mocking	Yes			more	14	less	15	lower	54	higher	6
		different	58	fewer	25	more	9	higher	10	lower	19
	No	true	41	true	60	true	74	true	34	true	73

Table 1: **Detailed mocking survey results.** All values are percentages of the 91 respondents. Levels of knowledge we considered to be unreasonable are marked in bold in the accuracy row.

	Attribute Count					
	0	1	2	3	4	5
Unreasonable Fear	48	33	16	2	0	0
Uncomfortable	24	19	18	16	12	11
Interested in changing	18	22	19	21	11	10

Table 2: **Summary of survey results.** Aggregates are shown for the three specific questions addressed in Section 5.1.2. All values are percentages.

We have instrumented changes to the Android platform that allows each of the above managers to communicate with the DataDecoy replay service. The managers now implement `Messenger` handlers to receive `Messages` from external channels. On construction of a manager, it sends a `Message` to the replay service, notifying the service of its existence and establishing a bidirectional channel of communication. On receipt of a `Message` containing mocked data from the replay service, we can create actual `Location`, `SensorEvent`, and other objects from the serialized data packaged in the incoming `Message`. With these Java objects, we can now call the callback functions on listeners for the particular manager—for example, we can pass a newly constructed `Location` object to a registered `LocationListener` by calling its `onLocatoinChanged` method. To preserve the integrity of the mocking context, we also block real sensor events from being published to apps while replaying.

5 Evaluation

Our evaluation of DataDecoy has three aims. First, we establish that DataDecoy is desirable through a survey of 91 smartphone users (§5.1). Second, we show that DataDecoy works, by mocking several popular Android apps, and briefly discuss the overhead of the mocking process (§5.2). Finally, we demonstrate that DataDecoy is usable through a small user experiment where several volunteers were able to use DataDecoy to mock a pedometer app (§5.3).

5.1 Users Want to Use DataDecoy

Before beginning DataDecoy development, to gauge interest in mocking we distributed an IRB-approved survey to students, faculty and staff at our university. No incentives were provided for completing the survey, and all respondents were required to indicate consent before proceeding to the questions. Over four days, we recorded 91 responses.

5.1.1 Survey Questions

Table 6 summarizes the survey we distributed. It had three parts, each consisting of questions concerning five personal attributes: home location, weight, activity level, income level, and sociability. The goal of the first part was to assess how aware respondents were of the information smartphones apps could collect about them. Respondents were instructed to assume that the hypothetical app had been installed and granted the permissions it requested. The goal of the second part of the survey was to assess how comfortable respondents were with the data smartphone apps could collect about them. Finally, the third part assessed interest in mocking by determining how interested respondents were in misleading apps about the five attributes.

We intentionally chose a range of attributes. We considered home address and social network as straightforward to determine for an app with the right permissions, in this case the ability to track user's location (home address) or observe who they communicate with (social network). At the time we considered activity level to be more difficult to determine, although now that activity recognition has been integrated into widely-available libraries such as Google Play Services this may be much simpler to measure using the accelerometer. Finally, we chose two attributes that we were not sure could actually be determined by smartphones: income and weight.

Based on the capabilities of current smartphones we classified answers to the accuracy questions as either reasonable or unreasonable. As an example, we considered it unreasonable that an app could know a user's income to within $1 / year or their weight to within 1 lb. However, it is possible that by using the accelerometer and studying a user's gait their weight could be estimated, and the widespread adoption of smartphone-based payment systems such as Google Wallet along with socioeconomic map-matching may make income levels estimable soon. So even the answers we marked as unreasonable may not remain so for long.

5.1.2 *Survey Results*

Table 1 shows detailed results of our mocking survey. When analyzing the results, we were interested in three questions matching the three sections of our survey. First, how reasonable were respondents fears about information that apps might be able to determine? Second, how comfortable were they sharing data with apps? And finally, were respondents interested in modifying the data-driven impressions app might form of them? Table 2 reports aggregate results relevant to these three questions.

First, we found that respondents to be reasonably suspicious of what apps might know about them, with 52% indicating that an app might know at least one personal attribute to a level that we marked as unreasonable today but only 18% indicating that apps might know two attributes of unreasonable levels. The two unreasonable attributes most-frequently reported as knowable by respondents were their income to \$100 / year (24%), and the quantity and type of the exercise they engaged in (20%). Overall, users' intuition about what attributes were easy to determine and what were hard matched ours, reflected by the accuracy percentages in the table. No users thought an app would not be able to determine anything about their home address, whereas 39% didn't think an app could determine anything about their weight.

Second, our survey showed that many respondents were uncomfortable with smartphones knowing these aspects of their personal lives. Only 24% were comfortable, defined as a score of 2 or above on the 1–5 scale, with all five attributes, and a majority (57%) were uncomfortable with two or more. Our results match the privacy concerns reported by smartphone users to other surveys [4]. Reported comfort levels on individual attributes were also interesting, with users seeming the least comfortable with smartphones knowing their home address—which is possible—and their income—which, at least today, may not be. Comfort levels regarding knowledge of a users social network were evenly distributed.

Finally, when asked about mocking, of the 91 users that completed the survey, 82% wanted to mock at least one attribute and 60% wanted to mock two, with mocking users requesting an average of 2.6 mocking attributes each. Interest in mocking different attributes was well distributed, with the percentage of mocking responses per attribute varying from a low of 26% for activity level to a high of 66% for income.

Unsurprisingly, users were most interested in mocking attributes that they were uncomfortable with their smartphone knowing, such as home address and income level. Surprisingly, most users seemed to want to appear to make *less* money than they actually do, which is not what we expected. We speculate that this may be because users believe that they will see fewer ads if advertisers believe that they are poor. In any case, it shows that it may be difficult to determine what changes users see as desirable.

App	Waze Social Maps & Traffic
Installs	10–50 million
Mocking Objective	Mock location
Trace Length	2 minutes 27 seconds
Trace Size	516 K
Location Mocks	117
Sensors Used	GPS, Gyroscope, Accelerometer
Sensor Mocks	2773
Wifi Scan Mocks	115
Cell Location Mocks	117
Video URL	`http://youtu.be/GIqXP6b769c`

Table 3: **Mocking Waze.** Details of our Waze mocking trace.

Overall, however, the results lead us to the conclusion that smartphone users:

- have reasonable expectations about what smartphone apps might be able to learn about them,
- are uncomfortable with apps knowing these things,
- and are interested in misleading apps.

5.2 DataDecoy Can Mock Smartphone Apps

Having shown that mocking is desirable, we continue by demonstrating that our DataDecoy prototype works by using it to mock three smartphone apps. In each case we describe the app, discuss why users might want to mock it, describe our specific mocking objective and whether it was achieved.

5.2.1 *Mocking Maps*

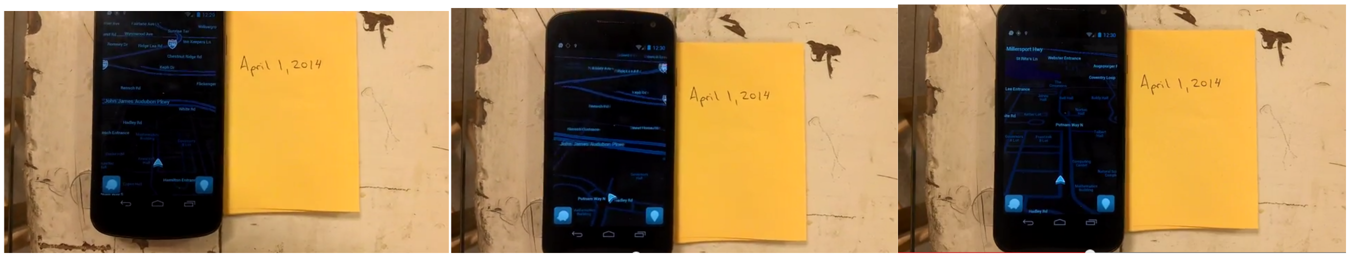
As a first example of DataDecoy in action, we mock the Waze maps app [8]. Waze describes itself as a community-based traffic and navigation app allowing "millions of drivers from across the globe joining forces to outsmart traffic, save time, gas money, and improve daily commuting for all".

Our objective in mocking Waze was to show that DataDecoy can provide fine-grained location to the many smartphone apps that use location to customize the user experience. Given the amount of information smartphone users' location can reveal about them, it is important for DataDecoy to effectively support location mocking. In our next mocking example we show the effect that mocked locations can have on an unsuspecting app.

To mock Waze, We first recorded a mocking trace of a walk around campus, described in more detail in Table 3. This included tracking location, connectivity, and sensor data from the gyroscope and accelerometer that Waze utilizes. We then placed the smartphone on a desk, initiated a replay session and launched the Waze app. Figure 2 shows three screenshots of Waze being mocked, showing that the users location is being updated to follow the trace despite the smartphone not moving. An anonymized video of our successful Waze mocking session is also available at `http://youtu.be/GIqXP6b769c`.

5.2.2 *Mocking checkins*

As an example of a more realistic mocking scenario, we used DataDecoy to mock the popular Facebook app [1]. Facebook is the world's largest social networking site and



(a) (b) (c)

Figure 2: **Mocking Waze.** The screenshots demonstrate that we were successfully able to mock the Waze maps app. As the mocking trace is replayed, the phone remains stationary on the desk, but Waze thinks that the user is walking around.

its Android app is quite popular, with the Play Store estimating between 500 million and 1 billion installs. People use Facebook to share content and stay in touch with friends.

Our objective in mocking Facebook was to initiate a check-in at a mocked location. Like FourSquare, the Facebook app contains a check-in feature allowing users to easily report that they are at a particular location, with check-ins being posted to their news feed and visible to their friends. Initiating mocked check-ins could be done for several reasons. Some Facebook users may want to appear more social than they actually are by initiating mocked check-ins at popular bars or clubs even on nights when they would prefer to stay home. Others may use mocked check-ins to compete for promotions provided by locations, or to compete for coveted status such as the "Mayorships" provided by FourSquare. Some FourSquare users have become so interested in earning the title of Mayor at a particular location that there is actually an entire website devoted to helping them[1].

To mock a Facebook check-in, we recorded a mocking trace of a walk to the nearby Starbucks, which included location, connectivity, and sensor data from the accelerometer. Because the trace was collected outdoors, no Wifi scan data was captured. We then placed the smartphone on a desk, initiated a replay session and launched the Facebook app. Figure 3 shows three screenshows of Facebook being mocked demonstrating that Facebook did allow us to check in at Starbucks at the end of the trace despite the smartphone not being located nearby. We did record a video of the mocking process, but due to the amount of personal information revealed by Facebook we were not able to anonymize it sufficiently.

5.2.3 *Mocking a game*

As a final mocking challenge, we attempted to use DataDecoy to mock an accelerometer-driven game. Fast Racing 3D is a car racing game available for Android [2]. Gaming is a popular activity on smartphones, with studies showing that 32 % of time spent on smartphones being devoted to game play [9]. While DataDecoy normally records sensor data aggressively during recording in order to ensure that it collects a superset of any information that could be requested during the mocking session, games provide a particular challenge for replay due to their aggressive use of high-rate sensor data—in this case, the accelerometer.

[1]`http://whenwillibemayor.com/`

App	Facebook
Installs	500–1,000 million
Mocking Objective	Check-in at mocked location.
Trace Length	1 minute 37 seconds
Trace Size	268 K
Location Mocks	106
Uses Sensors	GPS, Accelerometer
Sensor Mocks	2569
Wifi Scan Mocks	0
Cell Location Mocks	97
Video URL	(Removed for blind review.)

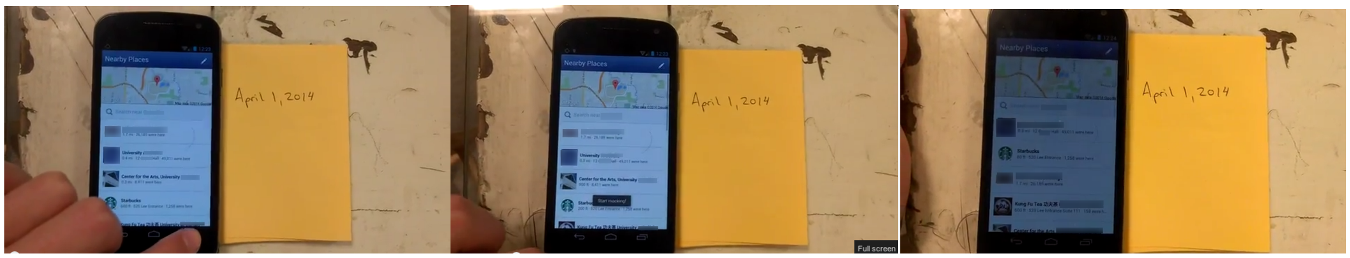
Table 4: **Mocking Facebook.** Details of our Facebook mocking trace.

Our objective in mocking Fast Racing was to use the mocked trace to control gameplay during the mocking session. Users may want to mock apps for several reasons: to avoid tedious replay of easier levels on apps that force players to begin again after failing more difficult challenges, or to improve their reputation through repetition when using apps that post scores to a public leaderboard.

To mock Fast Racing gameplay we recorded a mocking trace of a portion of a trip through one of the game's race courses. This included tracking location, connectivity, as well as sensor data from the accelerometer used to control the vehicle. We then placed the smartphone on a desk, launched the Fast Racing app, and initiated trace replay. Figure 4 shows three screenshots of Fast Racing being mocked demonstrating that the accelerometer data was able to control the Fast Racing vehicle. For this particular scenario, however, we found it difficult to trigger the mocking replay at precisely the correct moment, with the associated time delay causing the vehicle's path to eventually deviate from the original trace as the mocking session continued. An anonymized video of our semi-successful Fast Racing mocking session is also available at `http://youtu.be/8fAj5dYFwS0`.

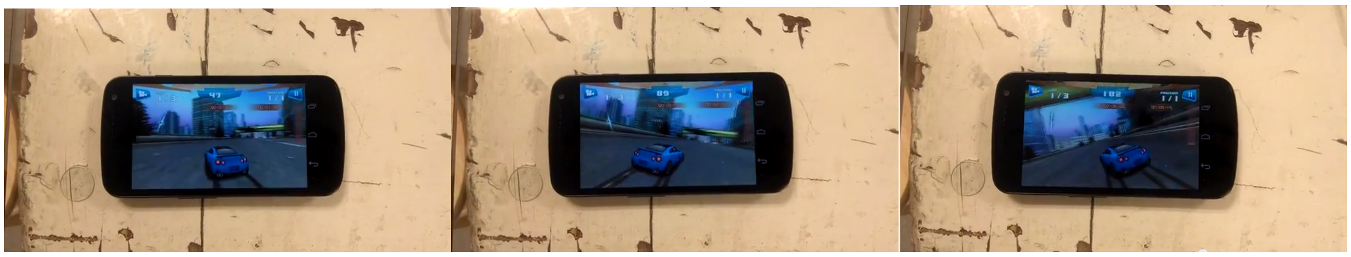
5.2.4 *Mocking overhead*

To address an inevitable concern about the mocking process, we measured the energy usage of DataDecoy while recording and replaying context traces. We measured the energy consumption of DataDecoy using the Monsoon Power



(a) (b) (c)

Figure 3: **Mocking Facebook.** The screenshots show that we were able to mock the Facebook app and initiate a check-in at Starbucks despite being a half-mile away.



(a) (b) (c)

Figure 4: **Mocking Fast Racing.** The screenshots show that we were able to mock the accelerometer-driven Fast Racing game to control the vehicle during the mocking session.

App	Fast Racing 3D
Installs	50–100 million
Mocking Objective	Control game play
Uses Sensors	Accelerometer
Trace Length	1 minute 35 seconds
Trace Size	296 K
Location Mocks	96
Uses Sensors	GPS, Accelerometer
Sensor Mocks	1363
Wifi Scan Mocks	96
Cell Location Mocks	96
Video URL	`http://youtu.be/8fAj5dYFwS0`

Table 5: **Mocking Fast Racing.** Details of our Fast Racing mocking trace.

Monitor [5] attached to our Galaxy Nexus smartphone. Because DataDecoy records certain types of data only when triggered by a location update, we mounted the entire setup on a cart so that the phone could be moved during the measurement of the trace collection process.

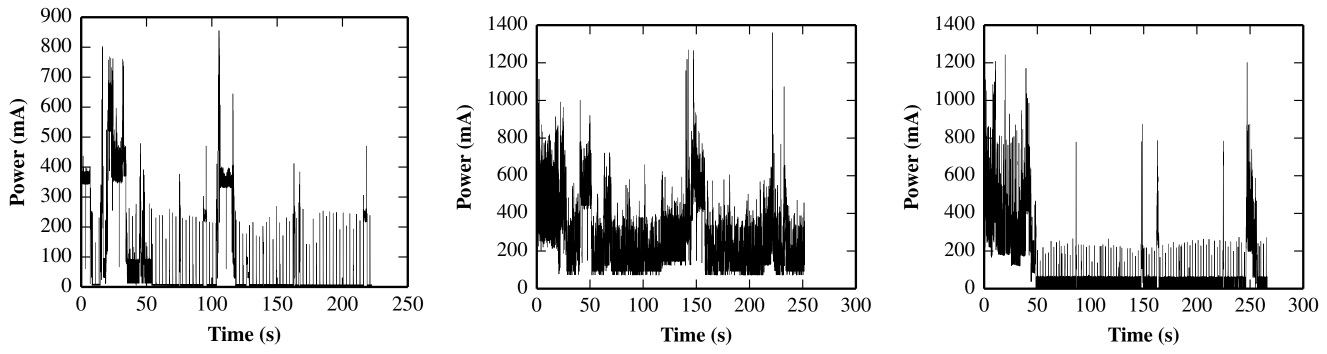
Figure 5 shows the power measured by the Monsoon. The first trace segment is a baseline showing idle power consumption. The second trace shows power consumption during the recording process, and the third shows replay. Because DataDecoy rapidly samples data from all available sensors during the trace recording process, this naturally leads to high energy usage, as shown in Figure 5b. During replay, however, mocking produces little overhead and energy usage is typical, as shown in Figure 5c. Because users must only record a trace one time but then can replay it multiple times, we consider the high overhead of trace recording to be acceptable. We are also designing further optimizations to reduce energy usage while replaying context traces, as because mocked data is being used real sensors—including power-hungry ones such as the GPS—can be disabled.

Mocking traces also consume disk space to store, with our experiments indicating that DataDecoy's traces consume roughly 3 K per second without removing repeated readings or performing compression. This means that 1 GB of disk space would allow a user to record almost 100 hours of mocking traces.

5.3 Users Can Use DataDecoy

With positive results from our previous survey and a prototype implemented, we wanted to gain a qualitative insight to future users of DataDecoy. Over the course of one day, we had 7 users use our most recent record-and-replay prototype. Due to physical limitations, we studied users individually with one Galaxy Nexus smartphone.

After receiving some background information on the project and instructions on how to use DataDecoy, users were given the smartphone with the goal of tricking an open-source Pedometer [6] that they were walking (and being active beings), while the phone was sitting on the desk in the lab. We wanted to receive feedback on the app, the idea, and the process, so we collected responses to the following questions after they saw DataDecoy in action: "Did DataDecoy mock the Pedometer?" and "Comments?". All users reacted positively, with one who "would love to use it for some other apps" and 42% of our users believe it has strong "potential".



(a) Idle power consumption. (b) Recording power consumption. (c) Replaying power consumption.

Figure 5: **DataDecoy Overhead.** Power usage is high during the record phase while DataDecoy rapidly gathers all data that might be needed during the mocking session. Replay (5c) power usage is typical of normal smartphone operation, with the screen going off after 40 s.

One user was so elated that he exclaimed "Much awesome, very app!".

The users found the app and process to be extremely transparent: all that was needed was to open the app, create an objective, hit record, and forget about it. In all cases, DataDecoy was able to trick the Pedometer into classifying the phone's action as walking—by showing steps increase—when the phone was really sitting idle on a desk.

6 Ethical and Practical Issues

Objective-based context mocking raises a set of unique ethical questions related to the relationship between smartphone users and apps, as well as a set of practical deployment challenges. We attempt to address both in this section. Even if not widely-deployed, we hope that the availability of this feature will help create a useful dialogue about smartphone data collection and user privacy.

6.1 Is Mocking Cheating?

A common reaction by many to mocking is to conflate it with cheating, given that it involves misleading apps about a users true nature. This response, however, begs the question: what are the rules of the game? Cheating must be defined with respect to an agreement, and we are not convinced that users have actually agreed to provide an unlimited amount of information about their personal lives to smartphone apps. While installing an app does involve allowing it to access certain information, we believe that it is reasonable for users to expect that apps request information required by the features they provide and use it only to provide those features. Unfortunately, particularly once the information leaves the device, users quickly lose control over how their data is used. Mocking can help return this control.

Another common objection is that the current smartphone app ecosystem depends on collecting user information in order to subsidize app development, most commonly by using personal information to embed targeted advertisements in apps, and that users benefit from lower app prices as a result. There are two problems with this argument. First, as mentioned earlier smartphone users have indicated on surveys that they are not comfortable with this model of subsidizing apps through personal data collection. Second, believing that this model is really a good deal for smartphone users requires them to trust the same companies that are actively trying to monetize this information. A sense that they are not receiving adequate compensation for personal information may drive expressed discomfort with this business model. In any case, because smartphone users have different privacy concerns and expectations, not every user should be required to trade their data for access to apps.

6.2 Is Mocking Safe?

A more serious concern with mocking concerns the effect it might have on apps, particularly ones that are health-related. If mocking confuses an app designed to remind a user to take a pill, it could have serious health consequences. Here it is important to distinguish between the possible side-effects of mocking on legitimate apps and the intentional effect of mocking on apps that the user is intending to mislead. For example, if a doctor asks a patient who wants to get fit to install a pedometer to help increase their activity level and the patient chooses to mislead this app with mocked activities, then the main problem is not really the mocking capability. If the user wants the app to help them become healthier, they will cooperate with it; if not, they can always refuse to be monitored altogether or find another doctor. Here mocking is not needed to provide users with control over their personal information.

To preserve the correct operation of safety-critical apps that might be confused by context mocking, DataDecoy already provides users the ability to allow certain apps to always receive real data. As we develop more experience with our DataDecoy prototype we will focus on refining this mechanism more carefully and ensuring that it is effective. Additional interaction with apps could be helpful. For example, future version of DataDecoy may allow apps to issue a request to not be mocked, which the user could approve or ignore, helping remind users to adjust mocking settings as new apps are installed.

6.3 Can Mocking Be Deployed?

DataDecoy requires platform support. Given the locked-down nature of major smartphone platforms today, this means that deploying DataDecoy requires the cooperation of the companies such as Google, Apple, and Microsoft that maintain the dominant three smartphone platforms available today[2]. Unfortunately, we expect that all smartphone platform providers have an interest in perpetuating the exposure of personal information to apps that DataDecoy is intended to frustrate, making them unwilling to implement mocking.

There are two potential ways around this roadblock. First, we will explore integrating mocking into popular third-party Android platform distributions such as CyanogenMod, which provides alternative ROMs to users interested in replacing their stock Android image by rooting their phone. While this community is small, they may be disproportionately interested in mocking given their willingness to void their warranties and the intentions of smartphone device manufactures. Even a small number of users using DataDecoy could lead the smartphone privacy conversation in the right direction. Second, at some point smartphones may be required to provide more configurability at the platform level to support apps, similar to the way that desktop operating systems allow apps to install device drivers. This feature would allow DataDecoy to make required platform changes.

6.4 What Effect Would Mocking Have?

Finally, we consider the effect that DataDecoy would have on smartphone data collection and privacy if deployed on a significant number of devices. First, we would expect to see an interest among app developers in deploying countermeasures to detect or eliminate mocked data. We have already discussed protecting the mocking device against attacks launched from apps running on the device and how the mocking context on Android can be secured through modifications to the Linux kernel. A more difficult or impossible set of attacks are launched by colluding with other devices or with surrounding infrastructure. Interdevice collusion is more feasible, since it could be launched by cooperating instances of the same app. In certain cases DataDecoy could mock cross-device interaction to fool the local app, or simply disable interfaces such as Bluetooth allowing device-to-device communication.

Collusion with the infrastructure would represent a more serious challenge to our approach. As an example, if mobile data networks began reporting smartphone user's location directly to app providers apps could use this information to pierce the mocking context by comparing the location being reported to them by DataDecoy to infrastructure-reported location. Usage statistics also visible to infrastructure providers could also be used to determine when DataDecoy was mocking a Wifi connection at particular location but actually running connections across the mobile data network. While this type of collusion is the most effective way to shut down our mocking approach, it would also represent an unprecedented level of cooperation between network providers and the companies selling apps and services. We anticipate that these types of agreements would be highly-unattractive to smartphone users already concerned about their privacy.

Our ultimate hope is developing objective-driven context mocking is to initiate a conversation about what our personal data is worth. Today, because smartphone users lack effective tools to control the data they provide to apps, they are effectively surrendering their personal information without receiving anything in return. So while this data is clearly worth something, as evidenced by advertisers scrambling to develop novel location-based analytics, as long as we give it away for free we will never know how much. If mocking causes apps to begin to be suspicious of the personal data they can collect, this may help make legitimate information about users valuable again.

7 Related Work

While DataDecoy is the first system to provide objective-driven context mocking, worries about smartphone app data collection have led to a number of previous efforts in this area. We divide related work into several categories. First, we examine mocking-based approaches similar to DataDecoy which deny apps access to data by returning fake or bogus values. Second, we look at novel ways to detect malicious apps or malicious app behavior through static analysis and information flow tracking. Finally, we examine proposals to improve Android's permission model to make it more effective and user-friendly.

7.1 Mocking Approaches

AppFence [16] is one example that uses data shadowing, blocking, and mocking to selectively deny data to apps on a per-permission basis. When apps request data the user has chosen to deny them, such as their phone number or email address, empty or fixed bogus values are returned. MockDroid [10] is another example of a similar system. Both these approaches focus on achieving privacy by limiting access to data, rather than achieving user objectives by manipulating data that apps do have access to, and we consider these efforts orthogonal to DataDecoy.

7.2 Record and Replay

DataDecoy utilizes the idea of record and replay as a first attempt of implementing objective-driven mocking on Android. Record and replay is not a new concept in general, but there has recently been a focus on instrumenting this type of procedure on mobile devices [14]. There are a few systems in existence, but the majority of research efforts for this type of system seems to be focused on developer testing and app performance. For instance, there is RERAN [15], which captures input events at the filesystem level for later programmatic replay; another example of such a testing system is VanarSera [19], which records event data then distributes it to multiple "monkeys" or slave devices for parallel testing; there is also AppInsight [20], an app-layer instrumentation that helps identify critical paths in user transactions on a per-app basis. Despite this focus, there is interest in using record and replay systems to bring permission issues to light [21].

7.3 Permissions Improvements

A number of previous systems have explored ways to overcome Android's "take-it-or-leave-it" permission model

[2]Because their platforms are closed-source, Apple and Microsoft would have to implement this feature themselves. We are preparing a patch for Google's Android Open Source Project.

by allowing apps to run but selectively denying them access to information that they think they can access. There are other approaches out there to try to amend this model by allowing users to decide which permissions to accept [17], but users could accidentally reduce app functionality by rejecting a particular permission. Giving users the ability to selectively accept permissions does not fix the current permissions model because many apps request permissions for private data that are necessary only to third-party components, like advertising libraries [18]. To thwart this, AdDroid separates the requested permissions by advertiser-required and application-required.

8 Future Work

By enabling objective-driven context mocking, DataDecoy opens up many new directions for future work.

8.1 Securing the Mocking Context

While we have demonstrated that DataDecoy can effectively mock unsuspecting smartphone apps, future apps may attempt to defeat user mocking by performing more validation of the information they receive from smartphone platforms. As described previously in Section 4, because it is implemented at the platform layer our current prototype cannot defend against even simple attacks that bypass Android and read data directly from Linux. Our next step is develop a set of mocking patches allowing mocking to be implemented in Linux, which will address this issue.

More work is needed, however, to determine the best approaches to preventing apps from observing differences between the mocked context and the real environment. As an example, while it is simple to make a faster network device (Wifi) look like a slower one (3G) through bandwidth throttling, the reverse is not possible. However, one approach may be to mock conditions associated with a congested or low-signal strength Wifi link when the real context is using 3G, or mislead the app into thinking that its bandwidth is being throttled by the Android platform.

8.2 Per-App Mocking

At the moment DataDecoy feeds mocked data to all apps, which can create undesirable consequences in certain cases. As an example, if a user wants to perform mocking but also use their smartphone for navigation during the mocking session, this is currently impossible since the navigation app will be receiving incorrect locations from DataDecoy. We are working on improving the platform support DataDecoy requires to allow user-specified apps to pierce the mocking context and receive real data. Once able to perform mocking on a per-app basis, DataDecoy could mock several apps simultaneously using different mocking traces designed to achieve different objectives.

8.3 White Hat Data Analytics

Currently the process of linking user objectives with mocking traces is qualitative in nature: DataDecoy can suggest that a user interested in seeming more fit take a walk. To make this process more quantitative, we are augmenting DataDecoy with a library of data analysis algorithms that attempt to replicate the approaches taken by companies mining data, such as location traces, provided by smartphone apps. We refer to these open-source algorithms as *white hat data analytics*, reflecting their intention to help individual users in the name of privacy, rather than companies in the name of profit. Running these algorithms on the phone, rather than in the cloud, preserves users' privacy. White hat data analytics will play two roles in future versions of DataDecoy. First, they will reveal to users what their smartphone data might reveal about them helping them choose what attributes to mock. Second, they provide a way of quantitative evaluating mocking traces, allowing Bob to measure the impact his walk will have on his activity level and compare it with, for example, a bike ride.

8.4 Mocking Libraries and Synthesis

Mocking performed by DataDecoy is currently limited to repeating actions that the user has previously recorded. While this is a powerful primitive, it is also limited, including in ways that complicate DataDecoy's task of preserving spatial continuity as described previously. However, as the size of DataDecoy's trace library grows it becomes more likely that our system *can* synthesize new mocking activities by combining parts of earlier traces. Future versions of DataDecoy will facilitate this process by performing location-driven and battery-permitting background recording of context traces as users go about their daily lives. It is also possible that DataDecoy could eventually draw on shared libraries of context information to synthesize activities even falling outside of a user's own library. If another DataDecoy user has mapped the walking paths, Wifi locations, visible cell towers, and other shared context information at a particular location, then DataDecoy may be able to merge that data into an individual trace of a user walking to allow them to take a walk in a place they have never visited.

8.5 Mock to Test

Finally, we also note that mocking traces may also be valuable to smartphone developers interested in testing how their app performs under varying conditions. By adding support to the Android emulator for reading mocking traces, the user of a new mapping client could experience exactly how their app would perform for another user navigating the streets of a city far away. This feature is similar to Android's current ability to allow apps to use mocked locations, but by also mocking all other location-varying aspects of the device's context DataDecoy would provide more realism.

9 Conclusion

In this paper, we have introduced the notion of objective-driven context mocking to help users protect their digital personas on their mobile devices. We have discussed DataDecoy, a prototype implementing a record and replay system to evaluate the idea of objective-driven mocking and mocking sensor data in general. Through our case studies, we have found the prototype was able to mislead multiple apps with minimal overhead. Not only did we find that this was technically feasible, but users are excited about the idea of being able to mock their data and want to be able to use DataDecoy in the future to protect their digital identities.

What can data collected by your smartphone reveal about you?

The questions below assess how much you think your phone is able to determine about you without asking. All the questions assume that you have installed the application and granted it the permissions it requested.

- Without asking, what could an application determine about your yearly income?
  - An application could predict my income exactly, to within \$1 / year.
  - An application could predict my income to within \$100 / year.
  - An application could predict my income to within \$10,000 / year.
  - An application could not determine anything about my income level.
- … your weight?
- … your social network?
- … your activity level?
- … where you live?

What are you comfortable with applications knowing about you?

The questions below assess how comfortable you are with smartphone application being able to determine the same things about you we asked about in the first section. **Please indicate your comfort level between not comfortable at all (1) and completely comfortable (5).**

- How comfortable are you with smartphone applications knowing your yearly income?
- … your weight?
- … about your social network?
- … your activity level?
- … where you live?

Would you like to alter what your smartphone knows about you?

These questions assess your interest in altering what your smartphone knows about you. Assume that a system exists that would allow you to change the qualities as indicated by the questions.

- If a smartphone application could accurately determine my income level
  - I would like to appear to have a lower yearly income than I actually do.
  - I would like to appear to have a higher yearly income than I actually do.
  - I am comfortable revealing my true income level to the application.
- … my weight
- … my social network
- … my activity level
- … where I live

Table 6: **Mocking survey questions.** Respondents were asked three groups of questions about five aspects of their personal lives their smartphone could observe. For each group one sample question and answers is shown.

10 References

[1] Facebook. `https://play.google.com/store/apps/details?id=com.facebook.katana`.

[2] Fast racing 3d. `https://play.google.com/store/apps/details?id=com.julian.fastracing`.

[3] Google wallet. `http://www.google.com/wallet/`.

[4] Mobile Privacy: A User's Perspective. `http://www.truste.com/why_TRUSTe_privacy_services/harris-mobile-survey/`.

[5] Monsoon power monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[6] Pedometer. `http://code.google.com/p/pedometer/`.

[7] Samsung galaxy nexus. `http://en.wikipedia.org/wiki/Galaxy_Nexus`.

[8] Waze social gps maps and traffic. `https://play.google.com/store/apps/details?id=com.waze`.

[9] What are smartphones for? apps and gaming. `http://www.themalaymailonline.com/tech-gadgets/article/what-are-smartphones-for-apps-and-gaming`.

[10] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: trading privacy for application functionality on smartphones. In Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM.

[11] 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 9th USENIX conference on Operating systems design and implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[12] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[13] 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 Eighth Symposium on Usable Privacy and Security, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[14] J. Flinn and Z. M. Mao. Can deterministic replay be an enabling tool for mobile computing? In Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, HotMobile '11, pages 84–89, New York, NY, USA, 2011. ACM.

[15] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: timing- and touch-sensitive record and replay for android. In Software Engineering (ICSE), 2013 35th International Conference on, pages 72–81. IEEE, 2013.

[16] 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 18th ACM conference on Computer and communications security, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM.

[17] M. Nauman, S. Khan, and X. Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.

[18] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: Privilege separation for applications and advertisers in android. In Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS '12, pages 71–72, New York, NY, USA, 2012. ACM.

[19] L. Ravindranath, S. Nath, J. Padhye, and H. Balakrishnan. Automatic and Scalable Fault Detection for Mobile Applications. In Proceedings of the 13th International Conference on Mobile Systems, Applications, and Services (MobiSys), 2014.

[20] L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan, I. Obermiller, and S. Shayandeh. Appinsight: Mobile app performance monitoring in the wild. In Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12, pages 107–120, Berkeley, CA, USA, 2012. USENIX Association.

[21] L. Yang, N. Boushehrinejadmoradi, P. Roy, V. Ganapathy, and L. Iftode. Short paper: Enhancing users' comprehension of android permissions. In Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '12, pages 21–26, New York, NY, USA, 2012. ACM.