# Controlling Smartphone User Privacy via Objective-driven Context Mocking

Nick DiRienzo and Geoffrey Challen
Department of Computer Science and Engineering, University at Buffalo
{nvdirien,challen}@buffalo.edu

***Abstract*—Smartphones represent the most serious threat to user privacy of any widely-deployed computing technology because these devices are always on and always connected, making them the perfect candidate to know most about the owner. 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 of their data; for example, 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). As a result, smartphone users find themselves forced to make burdensome and error-prone trade-offs between app functionality and privacy. To combat this, we propose a new approach called PocketMocker. By allowing substitution of real data streams with artificial or *mocked* data, PocketMocker allows users to manipulate impressions of their behavior in well-defined ways, such as appearing more fit, more social, or more on-time than they actually are. Instead of focusing on privacy, we explore providing users with better management of their smartphone-derived digital identities. We discuss the design of PocketMocker, which uses user-initiated context trace recording and replay to enable objective-driven context mocking. Our evaluation shows that users want to use PocketMocker, that PocketMocker can mock popular smartphone apps, and that PocketMocker is usable.**

## I. Introduction

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 on social networking sites can already make themselves look more attractive or seem more interesting by actively selecting the pictures they upload or the activities they share. 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.

This loss of privacy is not ignored by smartphone platforms. These platforms try to protect users by having apps request sensitive user data through permission mechanisms. Unfortunately, the currently used model in Android— where users have to either install the app with all of its permissions or not install at all—leave users exposed to two problems: apps tend to request more than required [1], [2] and users do not understand why apps request particular permissions [3]. An alternative approach [4] allows users to decide which permissions to grant, but users have to choose between app functionality and their privacy because they are unaware how apps may behave if a particular permission request is rejected. This leads to users accepting all permissions for the apps they wish to install despite not fully knowing how that data may be used. Overall, permissions are not user-centric as apps request access to data for both legitimate and illegitimate actions: a navigation app may use location data for both navigating and for advertising. It is unreasonable to expect users 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 an 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 PocketMocker, a system enabling objective-driven context mocking. PocketMocker'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.
3) We evaluate PocketMocker and show it to be both desirable and effective. Field testing of a PocketMocker prototype demonstrates that it can successfully mock several popular apps and users are interested in using it.

The rest of our paper is structured as follows. After motivating PocketMocker with several examples in Section II, we describe PocketMocker's design and implementation in Sections III and IV. Section V evaluates our PocketMocker prototype, showing both that PocketMocker works and that smartphone users are interested in using the capabilities it provides. We review related work in Section VI, discuss future plans in Section VII, and conclude in Section VIII.

MobiCASE 2014, November 06-07, Austin, United States

DOI 10.4108/icst.mobicase.2014.257783

## II. Motivation

Second only to battery life, recent studies show that privacy is one of today's smartphone users' top concerns with their devices [5]. Despite the high demand for free ad-driven apps, 43% of users are *not* willing to share personal information with a company in exchange for a free app. Fortunately, smartphone user privacy receives significant attention from academia and industry. Protecting users' privacy is a worthwhile goal, but ironically, we believe 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 learn about me? Is all of the data this app collects really necessary? Is this app using my data for legitimate reasons? Companies are spending billions of dollars to better profile their users through data analysis algorithms and few, if any, have a business interest in releasing how their algorithms work or what they know about their users. Even with new tools to determine what and how much data is collected on a smartphone, it is still unknown how that data is being used.

Obviously users can always choose not to use apps with which they feel uncomfortable, but that is not an ideal solution if they are interested in a particular app's functionality. There are efforts out there to better protect users through safer app marketplaces by preventing distribution of malicious apps that only want to misuse the permission-protected data, but even non-malicious apps can be problematic. For the average user who reads email, browses the web, takes pictures and so on, their privacy may still be at risk to legitimate apps with legitimate data collection. At this point, users have two very unattractive options: removing useful apps or not using their smartphone at all.

To further investigate smartphone user thoughts on privacy, we distributed an IRB-approved survey—asking about their thoughts on smartphones knowing their address, friends, activity level, income and weight—to students, faculty and staff of the University at Buffalo; we discuss this survey in more detail in a separate publication [6]. 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. 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—such as income and weight—but only 18% indicating that apps might know two attributes of unreasonable levels. And 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. Our results show that users are concerned with what their smartphone may know about them, and users are interested in the ability to better control their private data on their devices.

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. PocketMocker provides smartphone users with the same control over their smartphone-derived digital identities.

### A. Mocking Scenarios and Types

Consider these four scenarios:

- Bob wants to appear more active. Instead of taking real walks, his smartphone can mock a walk while he sits at his desk during work hours.
- Alice wants to appear more healthy. While eating at a burger joint, her smartphone can mock a visit to the local vegetarian restaurant.
- Teenager Jerry is monitored by a smartphone app installed by his parents to ensure he is not out past curfew. He can stay out later-than-allowed with his friends, while his smartphone mocks his location to return home on time.
- Carol is monitored by a smartphone app setup by her employer to track attendance and working hours. Despite leaving for a latte with a friend, her smartphone can mock her still sitting at her desk.

These examples show that there is indeed a difference between *privacy* and *mocking*. The above objectives are concerned with privacy, but cannot be achieved by making data more private, thus this is where mocking can have a role to better protect users. In Jerry and Carol's case, the apps could be removed, but the installing party would notice; or in some instances, uninstalling an app would also mean forfeiting certain rewards for using it, such is the case with most "gamified" apps, assuming Bob and Alice are using such an app. In all cases, removing or disabling an app is not an attractive or possible option.

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

*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 visit to the healthy restaurant, it provides cover for her visit to the unhealthy restaurant.

In other cases, PocketMocker 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 PocketMocker 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.

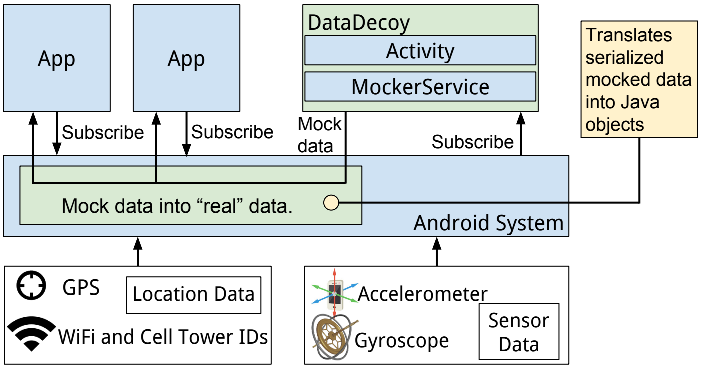


Fig. 1: **PocketMocker Design.** Details are specific to Android but would be similar on other platforms.

*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 PocketMocker to be able to dwell in a particular context while the user moves to another by looping a portion of a pre-recorded context.

## III. PocketMocker Design

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

### A. Overview

We offer the example of Bob from our earlier scenarios as an overview of how PocketMocker's components work together to deliver objective-driven context mocking. PocketMocker 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 PocketMocker.

Bob knows his objective: to appear more active. All he has to do now is collect some sample data so PocketMocker can inject mocked data to the apps on his smartphone when he wants to be active. First, he must record a mocking trace of his desired outcome—in this case, the action is taking a walk. During this phase, PocketMocker records all sensor data—such as GPS, cell tower metadata, visible WiFi access points and more—on the device to provide a concrete mocking context. All of this data combined is known as the *mocking trace*.

Once the trace is recorded, Bob can replay it as often as he likes. During the mocking process, PocketMocker 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 PocketMocker app displays a notification indicating that mocking is in progress and how much time remains before it finishes. After completion, PocketMocker 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 PocketMocker accomplishes these tasks and overcomes two consistency challenges below.

*1) Linking mocking traces and objectives:* To begin, PocketMocker 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, PocketMocker 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, PocketMocker 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 at the burger joint. PocketMocker'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.

*2) Collecting, storing, and replaying traces:* Second, PocketMocker must be able to collect, store and replay mocking traces. Trace collection is currently initiated by the PocketMocker app and performed entirely at the app level. To ensure that during mocking PocketMocker can return data from any source consistent with the mocked context, PocketMocker currently enables all sensors that could provide relevant information and samples them aggressively, storing timestamped data in a set of local databases. Unlike trace collection, mocking trace replay requires platform support. PocketMocker modifies the underlying smartphone platform to add an interface allowing it to inject mocked data. Once the user begins replaying the trace, PocketMocker 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) Initiating mocking sessions:* Third, PocketMocker's app helps the user remember to initiate mocking sessions. Each recorded trace can be annotated with a frequency which PocketMocker uses to help prompt the user to deploy the trace. For example, Bob may want to go for a walk daily, and by annotating the trace with his goal PocketMocker knows when to provide reminders.

### *B. 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 PocketMocker does not attempt to fully defend the mocking context from suspicious apps—we leave that challenge as future work. However, PocketMocker 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 PocketMocker masks differences between the mocking context and the real context. Then, we address spatial continuity, a specific consistency problem facing the PocketMocker system.

*1) Differences with the mocking context:* Here we examine specific differences between the mocking context and the real context and address how PocketMocker 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, PocketMocker 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, PocketMocker 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, PocketMocker 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, PocketMocker will establish connections over the available network but tell apps that they are connected over the mocked Wifi network. At present PocketMocker 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. There is also the problem of providing a mocked connection when no real connection exists. Even though this is impossible to practically accomplish, we can synchronize with the real context have PocketMocker simply return that there is no active data connection because networks naturally come and go, so this will not look suspicious to applications that attempt to circumvent the mocking context.

It is also important to point out that PocketMocker *does not mock*: 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 PocketMocker 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. Mocking battery levels represents another continuity challenge; at the time of entering or leaving the mocking context, the battery level will significantly increase or decrease to match the real or mocking context and apps can use this jump to detect a context switch has occurred, thus exposing a hole in the mocking context. Another problem with mocking battery levels is it creates a confusing user experience because the user then does not know the current status of the smartphone, which could have much harsher consequences, so PocketMocker uses the real battery levels always. Like battery levels, microphone readings also represent a continuity challenge which leaves a hole in the mocking context and we plan on addressing this current limitation of PocketMocker in the future.

*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, PocketMocker 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 VII how future version of PocketMocker 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, PocketMocker 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 PocketMocker 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, PocketMocker 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, PocketMocker continues to mock them at the mocking traces final location, using the lingering capability described next. This also allows PocketMocker 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.

### C. Lingering

In addition to the record-and-replay functionality we have already described, PocketMocker 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, PocketMocker 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, PocketMocker injects noise into the data returned during the lingering session, performing small changes to the reported location, sensor readings, scan results, and 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 PocketMocker will linger until instructed to proceed by the user. Once Alice is ready to leave the fast-food restaurant, she tells PocketMocker to proceed past the linger point she inserted into her trace of visiting the healthy restaurant. To conceal an undesirable activity, the PocketMocker 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 to her coffee break while appearing to remain at work.

Forward-shifting a trace is a three-step process. First, PocketMocker collects a small amount of context at the current location in order to linger and begins the lingering process. Second, PocketMocker 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, PocketMocker stops lingering and begins replaying the transition trace until the user reaches their current location.

## IV. Implementation

We implemented a PocketMocker prototype on Android 4.2.2 "Jelly Bean", Android being the only open-source smartphone platform permitting the platform modifications PocketMocker 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 V.

We considered implementing PocketMocker 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 PocketMocker prototype is implemented as a set of changes to Android. This is for two reasons. First, PocketMocker 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 PocketMocker 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.

The architecture of PocketMocker consists of two major components, 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 PocketMocker to notify user-installed apps of mocked location and sensory updates through their respective Java classes.

Our implementation of record and replay is user initiated at the app-level. When a user begins the recording process, PocketMocker aggressively logs device event data—including all sensors, GPS updates, cell tower updates and WiFi updates—to its SQLite datastore. By storing this data at the app-level, we can synchronize event broadcasts at the app-level by having the platform listen for updates from the PocketMocker replay service, a standard Android `Service`. The replay service works by reading events from the datastore in chronological order and notifying modified managers in the platform of the mocked data.

We have instrumented changes to the Android platform that allows each of the associated managers (`SensorManager`, `LocationManager`, `WifiManager` and `TelephonyManager`) to communicate with the PocketMocker replay service. The managers implement `Messenger` handlers to receive `Messages` from external channels and communicate with the replay service. On construction, a modified manager 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

| **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 I: **Details of Mocking Waze**

| **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** | `http://youtu.be/R8L6OV8hY2k` |

TABLE II: **Details of Mocking Facebook**

service, we call all callbacks registered with the manager, e.g. `LocationListener.onLocationChanged`, with the newly received mocked data. To preserve the integrity of the mocking context, real sensor events are prevented from being published to apps when PocketMocker is in replay-mode.

## V. Evaluation

We continue by demonstrating that our PocketMocker 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.

### A. *Mocking Maps*

As a first example of PocketMocker 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 PocketMocker 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 PocketMocker 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 I. 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. A video of our successful Waze mocking session is also available at `http://youtu.be/GIqXP6b769c`.

### B. *Mocking Check-ins*

As an example of a more realistic mocking scenario, we used PocketMocker to mock the popular Facebook app [9]. Facebook is the world's largest social networking site and 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. Facebook check-ins are shared with friends (and advertisers), so a user may want to create a fraudulent check-in for many reasons. It could be health-related, like our Alice, so different ads are displayed when the user visits the website, or it could be reputation-related as a user may want to appear more social than they really are.

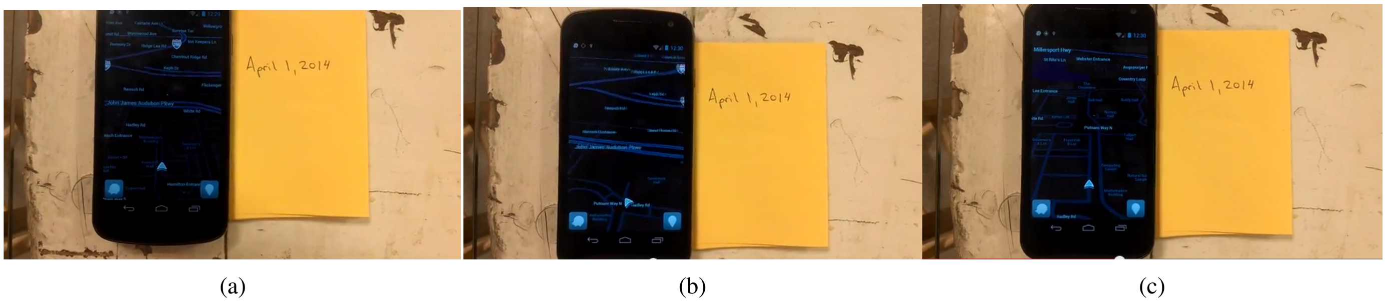
To mock a Facebook check-in, we recorded a mocking trace of a walk to the nearby Starbucks, which included location, connectivity, and sensor data from the accelerometer. Because the trace was collected outdoors, no Wifi scan data was captured. We then placed the smartphone on a desk, initiated a replay session and launched the Facebook app. Figure 3 shows three screenshots 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.

### C. *Mocking a Game*

As a final mocking challenge, we attempted to use PocketMocker to mock an accelerometer-driven game. Fast Racing 3D is a car racing game available for Android [10]. Gaming is a popular activity on smartphones, with studies showing that 32 % of time spent on smartphones being devoted to game play [11]. While PocketMocker 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.

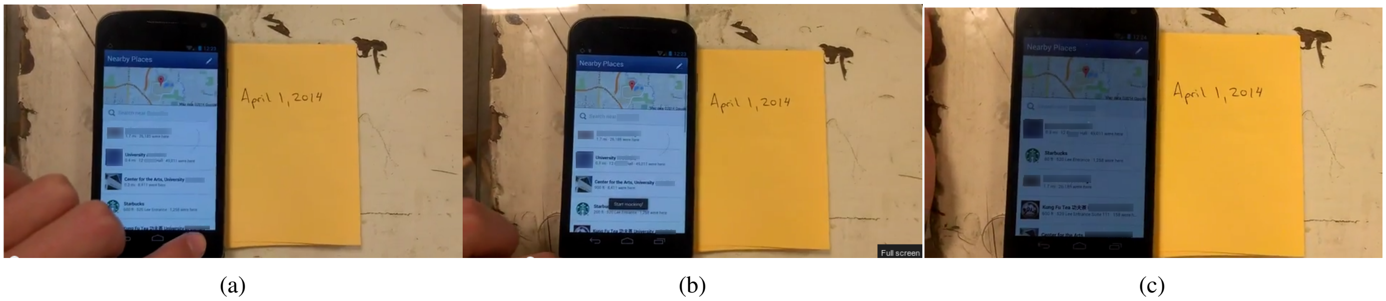
Our objective in mocking Fast Racing was to use the mocked trace to control gameplay during the mocking session. Users may want to mock apps for several reasons: to avoid tedious replay of easier levels on apps that force players to begin again after failing more difficult challenges, or to improve their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



(a) (b) (c)

Fig. 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.



(a) (b) (c)

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

| **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 III: **Details of Mocking Fast Racing**

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. A video of our semi-successful Fast Racing mocking session is also available at `http://youtu.be/8fAj5dYFwS0`.

### *D. Users Can Use PocketMocker*

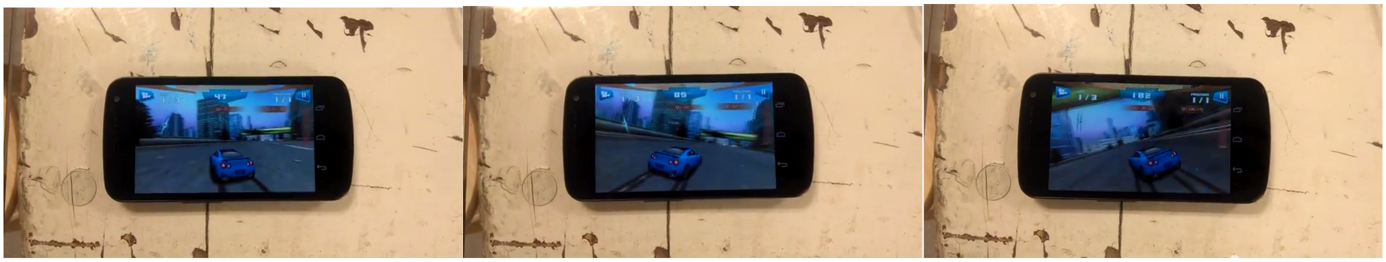
With a prototype implemented, we wanted to gain a qualitative insight to future users of PocketMocker. Over the course of one day, we had 7 users use our most recent record-and-replay prototype. Due to physical limitations, we studied users individually with one Galaxy Nexus smartphone.

After receiving some background information on the project and instructions on how to use PocketMocker, users were given the smartphone with the goal of tricking an open-source Pedometer [12] 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 PocketMocker in action: "Did PocketMocker 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".

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, PocketMocker 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.

## VI. Related Work

While PocketMocker 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. First, we focus on the approaches similar to PocketMocker which deny apps access to data by returning faked data. Then, 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 the Android's permission model to make it more effective and user-friendly.



(a)

(b)

(c)

Fig. 4: **Mocking Fast Racing.** The screenshots show that we were able to mock the accelerometer-driven Fast Racing game.

### A. *Mocking Approaches*

AppFence [13] 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 [14] 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.

### B. *Record and Replay*

PocketMocker 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 [15]. 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 [16], which captures input events at the filesystem level for later programmatic replay; another example of such a testing system is VanarSera [17], which records event data then distributes it to multiple "monkeys" or slave devices for parallel testing; there is also AppInsight [18], an app-layer instrumentation that helps identify critical paths in per-app user transactions. Despite this focus, there is interest in using record and replay systems to bring permission issues to light [19].

### C. *Permissions Improvements*

A number of previous systems have explored ways to overcome Android's "take-it-or-leave-it" permission model 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 [4], 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 [20]. To thwart this, AdDroid separates the requested permissions by advertiser-required and application-required.

## VII. Future Work

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

### A. *Improving security and user experience*

While we have demonstrated that PocketMocker can effectively mock unsuspecting smartphone apps, its security and user experience can still be further improved. Because of its implementation at the platform-level, there are possibilities for apps to know when they are being mocked. For example, they can read data directly from the kernel instead of calling through the Android platform. Our next step is to better protect PocketMocker by making changes at the kernel level. Another flaw with the current prototype is that mocking happens globally, meaning all user-installed apps are mocked when replaying mocked data. This can cause a problem if a user wants to use a user-installed navigator, but still want to mock another app while driving. We are working towards user-specified app mocking, so only apps selected by the user are mocked.

### B. *White hat data analytics*

Currently the process of linking user objectives with mocking traces is qualitative in nature: PocketMocker can suggest that a user interested in seeming more fit take a walk. We want to be able to provide users quantitative information as well, such as how apps are using their behavioral data to classify them. We will be augmenting PocketMocker with a library of open-source data analysis algorithms, so users can see how their data might be used to determine who they are. These open-source algorithms—or *white hat data analytics*—can also be used to help a user find out how well their mocking traces are working.

### C. *Sharing mocking traces*

PocketMocker is also limited by the device owner's previously recorded actions. While this is a powerful primitive, it is also limited, including in ways that complicate PocketMocker's task of preserving spatial continuity as described previously. We can grow their dataset in the background as they go about their daily routine by performing location-driven and battery-permitting recordings. There is also potential for users to share mocking traces, which would allow users to replay activity that they never performed, thus providing them even more options to shape their digital identity.

### *D. Mocking to test*

Finally, mocking can be used as a developer tool. By adding mocking support to Android, developers can test in a far more realistic way than ever before. For example, a developer building a new navigator would be able to provide all required sensor data and see how the app performs with real data while still sitting at a desk.

## VIII. 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 PocketMocker, 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 PocketMocker in the future to protect their digital identities.

## References

[1] 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, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[2] 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, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046779

[3] 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, ser. SOUPS '12. New York, NY, USA: ACM, 2012, pp. 3:1–3:14. [Online]. Available: http://doi.acm.org/10.1145/2335356.2335360

[4] 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, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 328–332. [Online]. Available: http://doi.acm.org/10.1145/1755688.1755732

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

[6] N. DiRienzo and G. Challen, "Should smartphone users mock apps?" in Proceedings of the 6th ACM Workshop on HotPlanet, ser. HotPlanet '14. New York, NY, USA: ACM, 2014.

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

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

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

[12] "Pedometer," http://code.google.com/p/pedometer/.

[13] 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, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 639–652. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046780

[14] 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, ser. HotMobile '11. New York, NY, USA: ACM, 2011, pp. 49–54. [Online]. Available: http://doi.acm.org/10.1145/2184489.2184500

[15] 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, ser. HotMobile '11. New York, NY, USA: ACM, 2011, pp. 84–89. [Online]. Available: http://doi.acm.org/10.1145/2184489.2184507

[16] 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. IEEE, 2013, pp. 72–81.

[17] 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.

[18] 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, ser. OSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 107–120. [Online]. Available: http://dl.acm.org/citation.cfm?id=2387880.2387891

[19] 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, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 21–26. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381940

[20] 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, ser. ASIACCS '12. New York, NY, USA: ACM, 2012, pp. 71–72. [Online]. Available: http://doi.acm.org/10.1145/2414456.2414498