Data Decoy: Useful Mocking of Location Data

Denise Blady
University at Buffalo
dcblady@buffalo.edu

Brian Rosenberg
University at Buffalo
bjr24@buffalo.edu

Eric Lehner
University at Buffalo
ericj@buffalo.edu

Guru Prasad
University at Buffalo
gurupras@buffalo.edu

ABSTRACT
Android as a smartphone platform has taken 31% of the U.S. market share, as of March 2011 [1]. As a comparison it has surpassed both BlackBerryôs and Appleôs market share [1]. Additionally, Android imposes an all or nothing permission model, where the user must either agree to all the requested permissions or give up the ability to install the application [2]. The problem with this permission model is that users may want to have selective control as to what permissions are allowed or denied, particularly when it comes to location data. In order to support that objective we have created a system called Data Decoy, which allows selective mocking of location data. In order to accomplish this we modified the Android platform, specifically the Location Manager. Our preliminary results show that Data Decoy provides effective mocking of location data, particularly with goal driven location traces. Overall we believe that Data Decoy is an effective tool for the privacy versus advertising war, allowing users to control their private information.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection ï *Access controls, Information flow controls.*

General Terms
Management, Performance, Design, Security, Human Factors.

Keywords
Advertising, Android, fine-grained access control, GPS, location, mocking, privacy, traces.

1. INTRODUCTION
As the smartphone platform with the biggest market share, Androidôs permission model affects more people than the other platforms in the market: Apple, BlackBerry, and recently Windows. What exactly does this mean for Android users? Androidôs permission model is simply put all or nothing [2]. A simple use case will demonstrate the permission model. Assume we have a user who wants to install ôMyCommuteô. When the user clicks ôInstallô they are given a list of permissions including System tools, Your location, Phone Calls, Network communication, and Your personal information. The ôYour locationô category gives further details including access to the users coarse and fine location. If the user wants this app they now have the option to ôAccept & downloadô, agreeing to all the permissions. It is the only way to install the application, clicking on permissions to accept only some is not an option. The effects of this policy are that the user must trust the developer in the appropriate use of their private information. Unfortunately not every developer fully understands the permissions required for their application, resulting in applications that are overprivileged [3]. This confusion often occurs with location permissions [3].

Location data is used for a number of reasons. Some reasons location data is often requested include: maps and giving directions, checking into nearby locations, advertising statistics, etc. Applications may legitimately need this information but there are also cases of applications asking for this data for no apparent reason. An application whose sole purpose is to display quotations has no need for GPS data. Advertising statistics is an interesting case. Why do advertisers need location data? Advertisers make use of user scores, which depends on a user profile storing demographics data such as location, to deliver targeted advertising content [4]. There is clearly a need for advertisers to request location data but users are left out of the loop when it comes to what data is collected to create a user profile [4].

The rest of the paper is as follows: Motivation, Design, Milestones, Preliminary Evaluation, Related Work, Future Work, and Conclusion.

2. MOTIVATION
The main motivation for Data Decoy is to allow users fine-grained control over their privacy. This not only bypasses the all or nothing Android permission model but it also allows users a say as to what location data is included in the user profiles used by advertisers. Another motivation for Data Decoy is the ability to use location traces for personal gain. Mocking location data should not only protect your privacy but it should provide users additional benefits, namely using location traces for a desired outcome. Overall, Data Decoy is designed to allow users full control over their privacy and it provides users ways to profit from location traces.

3. DESIGN
Data Decoyôs design is influenced by AppFence [5]. AppFence suggests two ways to protect userôs sensitive information - blocking data from leaving the device and sending false data [5].

Conference'12, Month 1ï 2, 2010, City, State, Country.

However, blocking data completely and sending unrealistic data breaks many applications. Therefore the only way to maintain usability of applications (while still protecting user's information) is to feed such applications with pseudo-realistic data gathered from the user's real information.

In designing Data Decoy, we wanted to minimize disruption and side-effects within the existing Android platform. The preexisting LocationManager code is designed to work with the LocationListener interface. We created the MockLocationListener class, which implements the LocationListener interface. This in turn allows us to leave the relationship between the Location Manager and the LocationListener interface untouched. This is highly advantageous, as almost all of Data Decoy's code is restricted to within our MockLocationListener class.

When an application registers itself with the LocationManager along with its LocationListener object, the LocationManager creates a MockLocationListener object that wraps the real LocationListener. This is then stored in the mListeners map. When an application receives a location update, the MockLocationListener's onLocationChanged method is invoked by the LocationManager. This method determines whether the application in question is to be mocked. If the application is to be mocked, the necessary mock location is computed and dispatched in place of the real location. If the application is not to be mocked, the real LocationListener object's onLocationChanged method is invoked instead.

Since we limit Data Decoy to work exclusively with location information, the operations that Data Decoy can perform are restricted to one of three types: Automatic Mocking, Manual Mocking, and Injection. Automatic Mocking is designed for users who wish to obscure their location without caring about what specific mock data their device is transmitting. Manual Mocking is designed for users who wish to obscure their location with specific mock data. Many use-cases support the cause for manual mocking. Applications such as Foursquare allow individuals to check-in at various points of interest. It also allows them to gain offers and coupons from the owners of these locations for being valued customers. Using Manual Mocking with applications allows users to report they are at a local restaurant or bar to receive promotional offers from these places. While the ethics behind this are ambiguous, we argue that any user who attempts to receive promotional offers in this way is a potential customer. Injection does not differ much from manual mocking. Both Automatic and Manual Mocking operate by modifying incoming location data; however, Injection is a proof-of-concept to show that it is possible to inject a location into the Android system. It should also be noted that injection is still in the early stages of development and is quite easy to detect.

4. MILESTONES

4.1 Milestone 1

For the first milestone we planned on mocking location completely, that is, to give applications a second, fake, latitude/longitude coordinate. Delivered with this milestone were two applications and a patch. The first application was the data decoy app, which displayed the latitude and longitude given by the Location manager as a simple way of demonstrating the mocked location's correctness. The second application was one with which to choose which applications the Location manager would mock. For our first milestone, this was not functional, but served as the basis of our front end application. The third and final deliverable was the Android patch, which properly mocked applications on the device it was applied to. All three of these were successfully completed, and it was after this milestone that we changed the purpose of our project from mocking multiple data types to mocking location with a purpose.

4.2 Milestone 2

The plan for the second milestone was to have mocking only affect select applications, and we had gotten a little farther with the GUI and proving the purposefulness of using certain locations to mock. The code that made this work was displayed in our powerpoint for our milestone 2 presentation, and we were able to test this code by simultaneously using Foursquare and google maps, with one showing the correct location and the other the mocked location. It was at this point that we started to accumulate Foursquare check-ins to test the acceptance of our "purposeful mocking" idea. To do so, we had numerous locations being randomly chosen by the underlying code. The GUI was also being improved at this point as the map API was being created to handle manual entry of GeoLocations.

4.3 Milestone 3

The third milestone plan was to have our final GUI, manual mocking, and location injection working, and to show the fruits of purposeful mocks. The final GUI displayed a MapView in the application which allowed users to enter locations to be used for manual mocking. To choose locations, users could either type in an address and/or by long touching on the location on the map. The locations on the map would create a list of latitudes and longitudes, or traces, for the underlying location manager to use. Whenever a manually mocked app were to call for location information, the code in the location manager would look at where the trace for the application was stored, and randomly select a location to return from the trace. This gave the user more of a direct control over which applications were given what locations. Location injection worked in concept, though failed when trying to inject into another application from our main application. We postulate this was due to Java's garbage collection. Finally, we displayed the end result of our quest to personally gain something from mocking a location. Through Foursquare, we were able to acquire 15 mayorships, 14 badges, and a variety of coupons close to our mocked location.

5. PRELIMINARY EVALUATION

In order to originally test the ability to mock locations with our patch, we first created a dummy app which would simply display the latitude and longitude of the user's position. This graduated to displaying the mocked location via google maps, and finally onto Foursquare. Foursquare is used to demonstrate automatic mocking and manual mocking. Injection is demonstrated through our test app.

5.1 Automatic Mocking

While using Foursquare we were able to show not only how location mocking worked, but how ill prepared companies are to deal with it. Foursquare mayorships are given to those who have

checked in the most to a specific location within the past 60 days. Over the course of a month, we checked in 216 times to 37 locations, earning 15 mayorships and 14 badges, as shown in Figure 1 and Figure 2 respectively. All of these check-ins are products of using mocked location data. Mayorships were not the only thing acquired, as Foursquare gives coupons to those who are in certain places. By mocking locations to places people would normally go, it allows one to see which coupons are offered in that particular location without having to actually go there.

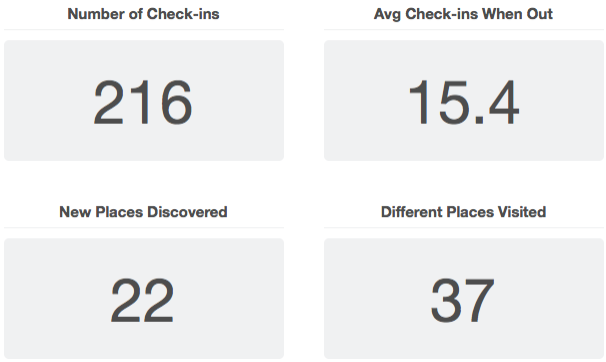

Figure 1. Foursquare statistics

5.2 Manual Mocking

Manual mocking has proved just as useful and robust, and was tested again with Foursquare. It was possible to check in at both the Museum of Modern Art (New York) and a coffee house on the Stanford campus (California) within seven minutes of each other. The Foursquare service accepted it without issue. Figure 3 shows that after changing the set of locations given to an application through manual mocking, the location changed instantly, as it should.

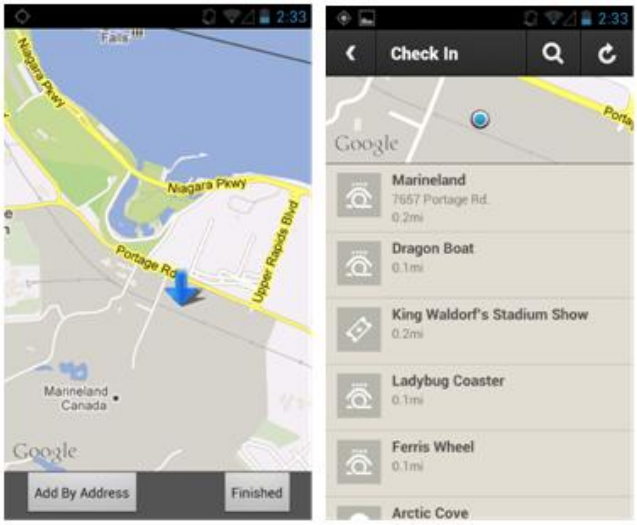

Figure 3. Manual Mocking with Foursquare

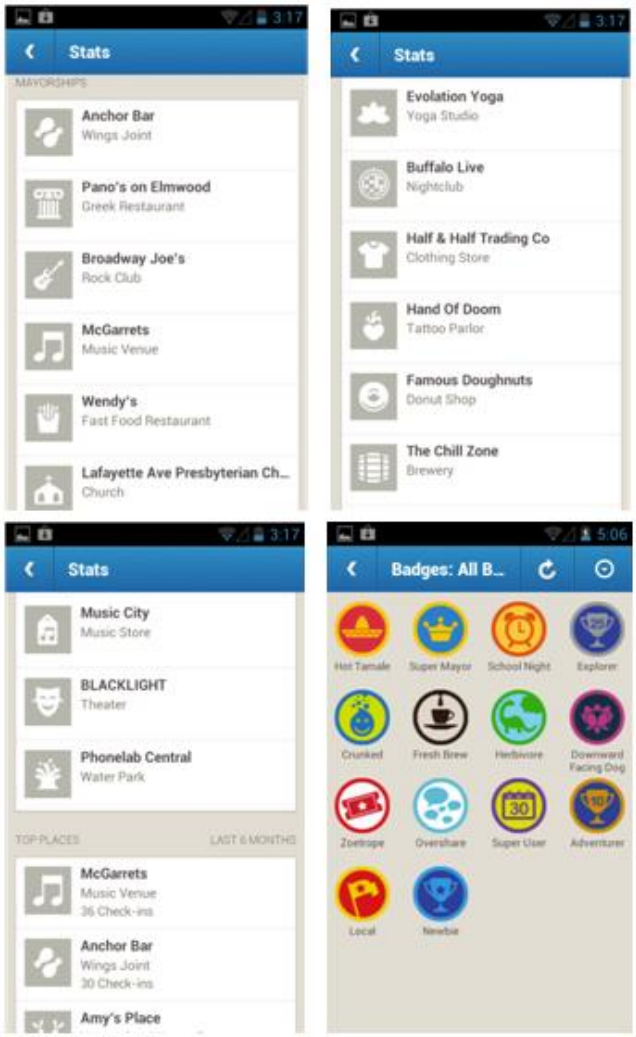

Figure 2. Foursquare Mayorships & Badges

5.3 Injection

Location injection was tested and proven feasible by a single application. When a button inside the application was pressed it would inject a location into the Location Manager. After this, it would display its current latitude and longitude given by its Location Manager, which would always be the injected location, shown in Figure 4 and 5. Unfortunately injection attempts failed when used outside of the application, possibly as a result of Javaôs garbage collection.

```
try {
    Method method = _locationManager.getClass().getDeclaredMethod("changeLocation", Stri
    String[] args = new String[] {"geolocationtest", "10 20", "gps"};    // CHANGE THIS
    int val = (Integer) method.invoke(_locationManager, (Object[]) args);
    if(val < 0)
        Log.d("MOD","changeLocation reflection call failed!");
    else
        Log.d("MOD", "changeLocation reflection succeeded!" + val);
```

Figure 4. Injection Code injecting GPS coordinates 10, 20

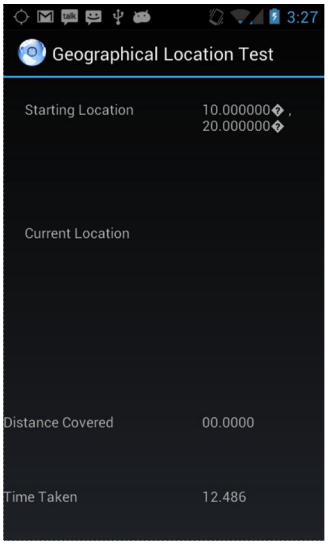

Figure 5. Location Mocking via Injection

6. RELATED WORK

Appfence uses data shadowing and exfiltration blocking, data mocking and data blocking respectively, to selectively mock and protect user privacy [5]. Data shadowing is implemented in a simple fashion such as returning empty values, fixed fake bogus alternatives (location, phone number), and in the case of the device ID (IMEI) a hash created from the real device id, application name, and a salt [5]. MockDroid is a system that allows real-time selective mocking of application permissions, where the requested resource is regarded as empty or unavailable whenever a user chooses to mock it [6]. Both of the above systems can provide selective mocking but it gives the user no control as to how they would like to mock their data and for what purpose.

Apex is a policy enforcement framework that allows users to selectively grant permissions to applications during runtime, but it is simply an extension of the permission framework currently used by Android [2]. Caché protects location data through grabbing real prior cached locations and providing a fuzzy location, in this case a general geographic region [7]. Both of these solutions fail to provide the ability to mock location data, limiting its usability.

Leontiadis et al. argues for a privacy model which strives to attain equilibrium between user privacy and profiling [8]. Their reasoning is that application markets are primarily driven by advertisements that rely on accurately profiling the user [8]. Leontiadis et al. achieve their goals through separate permissions for advertisements, and dynamic control over exposure of private data [8]. In our view user privacy is paramount. In the free application model users obtain applications by providing private information to advertisers. It is difficult to evaluate the worth of a userôs private information; therefore, since it is unclear as to whether this is a fair exchange we believe that these decisions are best left to the user. In addition we argue that the user should have absolute control over their personal information.

7. FUTURE WORK

In the future we would like to create a website similar to RetailMeNot.com. On the site people would upload, download, and vote on GPS traces that generated coupons, rather than coupon codes. In order to create this we would need a way to upload GPS traces that were found to generate coupons. We could use a special URI scheme for the links and make Data Decoy the application that handles that URI scheme. Then, when a user attempts to download a trace the Data Decoy app will receive the GPS trace.

Deciding which users to send coupons to can be thought of an optimization problem. Right now advertisers optimize for themselves. If users are able to swap GPS traces or coupons with each other, an application can optimize coupon delivery for the user. The best strategy for the advertiser is to send you coupons for places nearby that you wouldnôt otherwise stop. The best strategy for the users is to get coupons for places they were already planning on going. If we had an app that could analyze peopleôs ñstay pointsò, we could make the app send a storeôs coupons to your phone as soon as you walk in.

We would also like to expand Data Decoy to work on other private data types, such as contacts, phone number, IMEI, camera, calendar, etc. Many applications use third party ads and analytics packages. An interesting extension to Data Decoy would be the ability to give the ads and analytics code false data and the real code real data.

8. CONCLUSION

We presented a system called Data Decoy that mocks location data and provides the ability to use location traces. The interesting question now is what advertisers will do if a lot of people start using an application like Data Decoy. It is unlikely that advertisers will alter their behavior. RetailMeNot.com is very popular but coupon codes are still used. For more valuable coupons some kind of statistical analysis could be done to catch people mocking their data. Perhaps a protocol for verifiable location could be used.

9. REFERENCES

[1] King, Danny. "Google's Android Platform Overtakes BlackBerry in U.S. Popularity." DailyFinance.com. Dailyfinance.com, 03 July 2011. Web. 11 Dec. 2012.

[2] Nauman, Mohammad, Sohail Khan, and Xinwen Zhang. "Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints." Proceeding ASIACCS '10 Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (2010): 328-32. Print.

[3] Felt, Adrienne P., Erika Chen, Steve Hanna, Dawn Song, and David Wagner. "Android Permissions Demystified." CCSô11 (2011): n. pag. Print.

[4] Reznichenko, Alexey, Saikat Guha, and Paul Francis. "Auctions in Do-Not-Track Compliant Internet Advertising." CCSô11 (2011): n. pag. Print.

[5] Hornyack, Peter, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. "ñThese Arenôt the Droids

Youôre Looking Forò: Retrofitting Android to Protect Data from Imperious Applications." CCSô11 (2011): n. pag. Print.

[6] Beresford, Alastair R., Andrew Rice, Ripduman Sohan, and Nicholas Skehin. "MockDroid: Trading Privacy for Application Functionality on Smartphones." HotMobile ô11 (2011): n. pag. Print.

[7] Amini, Shahriyar, Janne Lindqvist, Jason Hong, Jialiu Lin, Eran Toch, and Norman Sadeh. "Caché: Caching Location-Enhanced Content to Improve User Privacy." ACM SIGMOBILE Mobile Computing and Communications Review 14.3 (2010): 19-21. Print.

[8] Leontiadis, Ilias, Christos Efstratiou, Marco Picone, and Cecilia Mascolo. "Donôt Kill My Ads! Balancing Privacy in an Ad-Supported Mobile Application Market." HotMobileô12 (2012): n. pag. Print.