6-2013

Metis: Mocking Data for Usability & Privacy

Brett Cooley
College of William and Mary

Metis: Mocking Data for Usability & Privacy

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science with Honors in Computer Science from The College of William & Mary

by

Brett Cooley

Accepted for: _____

_____
Haining Wang, Advisor

_____
Gang Zhou

_____
Ryan Vinroot

Williamsburg, Virginia
May 2, 2013

Abstract

As smartphone usage continues to grow, one of the most important unresolved issues is protecting the privacy of data, both stored by and generated from mobile devices. Various attempts at protection have been proposed, however, how to meet both the privacy and usability requirements well is still an open problem. This thesis presents a solution which provides privacy while minimizing the impact on usability. By providing fake information to unwanted but unavoidable requests for sensitive data or sensor readings, we allow applications to execute unhindered. Meanwhile, in doing so we also protect the user's privacy and offer a graceful degradation of service when mock data is given to the requester. We implement our design to work within existing and novel permission frameworks for the Android OS, and measure performance and usability impacts.

Acknowledgments

I would like to thank my advisor, Dr. Haining Wang, for his guidance throughout my undergraduate career and his unmatched patience. Without him, this honors thesis would not have been possible. I would also like to thank Dr. Gang Zhou and Dr. Ryan Vinroot for agreeing to serve on my honors committee. I am grateful to the Android Open Source Project, which provides well-maintained documentation and access to a world-class mobile operating system. My thanks go out to Yue Wang for collaborating on the initial design of this project. Finally, I would like to thank my wife for her unending moral support.

Contents

1 Introduction

Data privacy has become a major issue for users of mobile devices which allow interested parties like advertisement companies access to highly personal information. From location accurate within meters to the names, phone numbers, birthdays and more of personal contacts, there is an unprecedented amount of data about users available. To combat this, we provide users with a method of giving applications they do not trust fake, or "mock" data. This allows applications which expect their requests for data to be fulfilled by valid data to function correctly while protecting the user's real data from being accessed. This is useful because many applications will request data that is not central to the purpose of said application. Users find these applications useful, but would prefer not to give the application *carte blanche* access to their data. The Android Operating System (OS) allows users to review what data and services an application has access to before they install the application. However, the user must either allow the application full access to the data it requests, or abort the installation process. This all-or-nothing approach makes it impossible for users to allow partial access to applications. This is where Metis steps in, providing an intuitive, lightweight approach that bridges the gap between privacy and usability.

For example, users may want to install an Internet radio application which will tell them what their friends who also use this application are listening to. When installing this application, the users must allow the application access to their Facebook friends as well as their location. The users face a dilemma because while they want the application to be able to access their friend lists, they do not trust the application with their location data. Without Metis, this is an impossible goal to achieve. With Metis, the users can specify that their location should not be shared with untrusted applications, and install the Internet radio application. When the radio application goes to access the user's Facebook friends, it receives the user's real friend list. However, when the application tries to access the user's location to better target advertisements, it will receive a mock location, thereby not disclosing the user's real location and allowing the application to function as intended.

1.1 Background

Usage of smartphones has grown exponentially in recent years, and has already surpassed traditional desktop/laptop usage in some areas [16]. As specific mobile ecosystems (Android, iOS, Windows Phone, etc.) mature, users are doing an increasing proportion of their work on mobile devices. By moving to a more mobile-centric workflow, users are storing their data in more widely accessible places including on their mobile devices and on widely available servers or groups of servers ("the cloud"). These locations offer greater flexibility than more traditional locations, but with this flexibility comes an increased risk of data misuse or theft.

Misuse can occur when users' data is sold to partners of the storing company or used to target advertisements. Often this occurs without the user's explicit consent by requiring the user to agree to lengthy and verbose terms of service. Even when users read the terms of service, they are often confused by the legal wording and structure of such documents [1]. Furthermore, companies frequently change their terms of service without giving any notification to their existing users. Therefore, users are largely ignorant of how their data is handled and secured, leading to rampant misuse and overexposure of user data.

Data theft is equally troubling, as even if a company acts in good faith with regard to user data it can still be leaked to third parties. Many large online services have had their users' data stolen via simple misconfigurations in their storage infrastructure [23, 25] or compromised employee accounts [13,17]. Despite rather large and repeated incidents [13,23], users continue to use online services to store their data. Alternative companies [3] exist which provide much stronger guarantees of privacy and security and have perfect track records, yet are less popular. Clearly, users value usability among other traits more than they do privacy. However, users do value privacy, particularly when it comes to dealing with advertisers. In fact, users will opt for *less* usability to stop what they perceive as unauthorized data collection [2].

1.2 Motivation

As the previous section explains, data privacy is clearly a fundamental issue to many users of mobile devices. However, users are not willing to sacrifice usability to protect their data in most cases. In some ways, users view privacy and usability as mutually exclusive properties.

While every user has different reasons, the question of why user data should be considered private in the first place is important to examine. User data has clear value to companies [18], which sets a baseline by which its value may be judged. Further, United States Law provides legal protection of data users do not explicitly consent to transmit to the party who ends up using their data [4]. When user privacy is broken by a company, the potential monetary damages are severe [14]. There is ample evidence to support a user who expects its data to be private until explicitly given to a third party.

Given that data privacy is important, the need for measures which provide data privacy to be usable seems clear. Without usable means of protecting data privacy, users will forgo such protections in favor of less secure, more fluid options. This can be seen as a failure of technology to provide a viable solution to a reasonable need. In the next section, a survey of the current work in this field shows that no current system successfully fills both roles of protecting user data and being usable for the end user.

1.3 Outline

Section 2 discusses related work. Section 3 describes the design of Metis, including an architecture overview. Section 4 covers the details of our implementation and description of the additions to the Android framework. Section 5 presents an evaluation of both the performance and usability of our solution. Finally, we conclude in Section 6.

2 Related Work

The subject of data privacy has been a hot area of research recently. Others have looked at tracking data flows, expanding and augmenting permission-based systems, separating advertising from applications, and anonymizing user data for use by advertisers. As Wei *et al.* [27] demonstrated, the permission landscape in Android has grown over time, mostly

due to the addition of new kinds of hardware to smartphones. They also show that many applications are "over-privileged", meaning that they request more permissions than they need to function correctly. In a similar study, Felt *et al.* [8] programmatically detect over-privileged applications by analyzing which API calls the applications use, and mapping these to the permissions they require to function. They find that more than 33% of all applications are over-privileged.

2.1 Fine-Grained Authentication

This points to the need for some way to control applications that does not rely on the developer. The major attempts at such a system generally try to make authentication more fine-grained or to provide a mechanism for keeping private data private. Fine-grained authentication approaches attempt to increase the usability and adoption of authentication. In doing so, they help protect users from the potential for applications to misuse their data. There have been multiple attempts [7, 15, 19] at utilizing the sensors available on smartphones to determine if the owner of the phone is the one attempting to use it, and whether or not to require they authenticate themselves. Riva *et al.* [19] utilize location among other signals to detect if authentication is even necessary before the user can complete an action such as reading an email. Seifert *et al.* [15] develop a model of spheres wherein some applications can only be used when the phone is in a more secure sphere, such as at the owner's house or office. Researchers at Microsoft have explored [7] how users share their mobile devices, and experimented [9] with a guest mode where only applications without access to sensitive information can be used to facilitate device usage by multiple, untrusted guests.

Metis uses a similar approach as these efforts, but applies it to the Android permission model instead of authentication outright. By allowing fine-grained control over permissions, we keep authentication simple and predictable, and make the user interaction with our system a one-time configuration. With fine-grained authentication users may not be able to anticipate when their phones will require them to authenticate, leading to frustration when authentication is required at an inopportune moment.

2.2 Data Privacy

Instead of managing access to the device and applications, many have attempted to simply secure important data so that it cannot be misused by applications, malicious or otherwise. One of the earliest and most well-known attempts at controlling the data directly is TaintDroid [26]. TaintDroid is a system to monitor the flow of sensitive data throughout the Android system, and alert the user to data leakage. While being well designed and only modestly impacting performance, TaintDroid requires too much from the end user to be considered usable by a non-technical smartphone user. Metis takes a simpler approach, which increases usability at the expense of not having the insight into the flow of sensitive data like TaintDroid.

Pearce *et al.* [22] found that over 34% of applications request location data purely for a bundled advertising library. With such a high number of applications utilizing potentially invasive ads, many researchers have studied systems which would decouple application code from advertising code. AdSplit [24] complete separates the two, placing advertising libraries in their own sandbox. This allows them to work with the permission system to grant the application and advertisements permissions separately. AdDroid [22] provides an API and new permissions for advertisers built into the Android platform. In a slightly different vein, Leontiadis *et al.* [12] also separate the advertising networks from applications. However, being sensitive to the revenue model of in-application adverts, they allow for a variable amount of data to be sent to advertisers based on how frequently the user engages with the adverts. Tackling the issue at a different level, Haddadi *et al.* [11] propose a system which allows for anonymous data to be collected directly by the advertisers.

There have been other attempts at providing alternate data to applications. Hornyack *et al.* [21] provide "shadow" data, which is simply fake or empty data to applications which request access to it. They also allow for data to be marked as local-only data, meaning that any attempt to transfer this data over a network is blocked, using TaintDroid to detect such an attempt. These policies are applied to all applications on the device. In contrast, Metis provides a similar form of data masking, but does so only to applications which have been considered untrusted. This allows for critical applications like messaging or

direction applications to function correctly while still protecting user data from non-critical applications like games. In additioin, Metis allows for third-party providers to generate the mock data needed for untrusted applications. This is more modular than the approach of hard coding values into the OS and associated framework, and it can easily allow for realistic looking data in the event when application developers begin to verify the authenticity of the data they receive. Finally, while the idea of local-only data is helpful in protecting data privacy, it is not a guarantee that the data cannot be exfiltrated. A determined application can copy the data in ways which TaintDroid cannot detect or track, and then continue with exfiltration. Since Metis does not attempt to provide a comparable service, it does not suffer from the same issue.

Another effort to provide fake data to applications is shown by Zhou *et al.* [28]. Unlike the previous system, this one is highly configurable. The user may, for every application, view the permissions that the application was granted at install time. For each permission, the user can then select one of four modes: `trusted`, `anonymous`, `bogus`, or `empty`. The `anonymous` mode attempts to provide useful but unidentifiable data whereas the `bogus` mode simply returns some valid data. The other two modes function as expected, with `trusted` providing the real data and `empty` returning nothing. While this design provides a higher level of flexibility than Metis, setting modes for individual permissions on every application is unwieldy. Users are both ignorant of how to configure systems securely and unwilling to spend long periods of time to do so. Also, users generally are either fine with giving an application their data, or they are not. This binary preference lends itself to a binary model as opposed to the four state model used by these researchers.

2.3 Android Mock Package

Android contains a set of mock classes [6] intended for testing applications which rely on non-trivial amounts of real data such as a populated database or dynamically loaded resources. These classes can be extended by an application to mimic the function of a component without requiring that the mocked component actually be fully functioning or have useful data. While there is a superficial similarity between this form of mocking and ours, the intended use cases are very different. Android's `test.mock` package is to

be used for testing purposes while our mocking is intended to protect the privacy of user data. The provided classes are private components of applications, and cannot normally be accessed by other apps on the same device. Even if they are opened for public use, they cannot force other applications to use them instead of the associated real providers. Our approach transparently routes all desired requests to a mock provided regardless of application preference.

2.3.1 Mock Location

Outside of the mock classes provided in the `test.mock` package, Android has a built-in capacity for mock locations [20]. This allows any application with permission to respond to location requests as if it were a real provider. While this is closer to our approach, there are several problems making this not a robust solution. Firstly, the mock location is reported as GPS coordinates. This means that an application could ignore updates from any provider with the GPS type, and still get real location information. Even worse, an application could easily detect the use of a mock location by checking both GPS updates and a more coarse location provider like the network or Wi-Fi, and note the discrepancies. In contrast, our approach is consistent across any location provider, and cannot be evaded as it intercepts all location requests.

3 Solution Design

We now introduce Metis, our solution to the data privacy problems inherent in the Android OS. Metis provides effective data privacy while minimizing the effect on application usability. It ensures data privacy by flagging configurable portions of user data (with reasonable defaults) as private. When an application is installed, the user classifies it as trusted or untrusted. Trusted applications are allowed full access to any data for which they request permission. Untrusted applications are not allowed to access private data, instead being given "mock" data when they request private information. This substitution is completely transparent to the application, which allows for the application to continue to function while maintaining data privacy for the user. Metis achieves this goal through a combination of

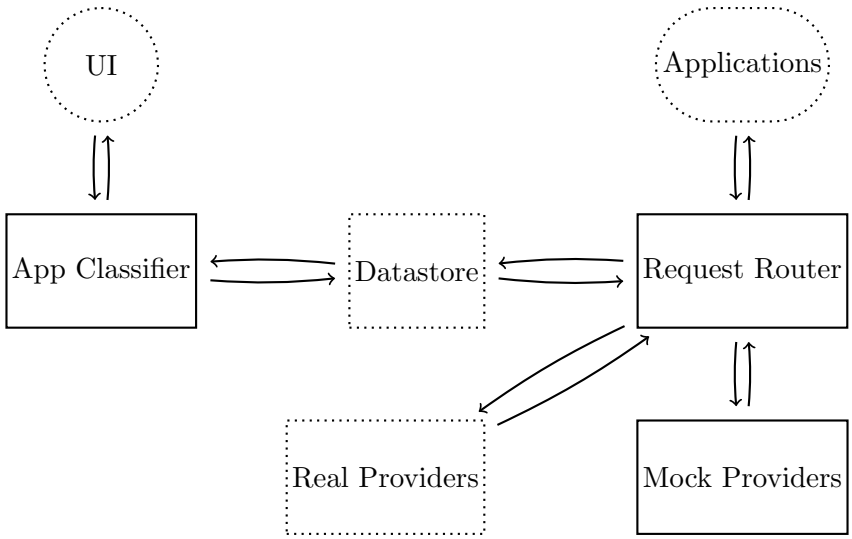

Figure 1: Metis architecture diagram

app classification, request routing, and mock providers.

3.1 Architecture

Metis is built around three loosely coupled modules, as shown in Figure 1. The user interacts with a UI to classify applications, which are stored to a persistent datastore. As applications send requests for user data, the router intercepts them, and checks with the datastore if the requesting application has permission to access the requested data. If so, the router passes the request on to the real provider. If not, the request is sent to the corresponding mock provider. Regardless of which provider handles the request, the data is then returned to the application.

3.1.1 App Classification

One of the key features of Metis is its simplicity of use. At install time, Metis will check the application's permissions against a list of permissions which are considered dangerous enough to leak private data. This list can be modified by the user, but is shipped with a reasonable default set of dangerous permissions. If the application being installed requests one or more of the dangerous permissions, the user will be asked if the application is trusted or untrusted. Trusted applications are allowed full access to any data they request

permission for whereas untrusted applications are restricted from data which requires a dangerous permission to access. The classifier stores the application's package name in a persistent datastore, along with whether it is trusted or not. This list of which applications are trusted can also be edited after install time if the user changes its mind.

Since the major purpose of the app classifier is to present the user with an intuitive interface, there is little actual complexity involved in writing such a classifier. While the classifier must be trusted by the user to be non-malicious, the runtime needs no special permissions beyond the ability to communicate with the datastore. Therefore, the classifier can sit in user-space. In order to ensure the integrity of the classifier, implementations may use tools specific to the implementation target platform. We utilize one such property within the Android framework, and discuss this more in Section 4.

3.1.2 Request Routing

When an application requests user data outside of its own control, the request is first passed to the router. The router checks if the permission associated with the request is considered dangerous. If not, the request is forwarded to its normal destination unmodified. Similarly, if the application making the request is trusted, the request is also forwarded. In the case that both the request is considered dangerous and the application is untrusted, the router modifies the request, instead sending it to the associated mock provider, which will return the mock data. All of this data is stored in the datastore, including which mock providers can service which kinds of data requests.

Intercepting requests for data needs to be performant as such requests are very common in most applications. In order to provide such performance along with a router that cannot be bypassed by malicious applications, the router is run as a privileged service. It is designed such that modifications to the router itself should not be needed for either changes to the user interface or changes to how the mock data is generated and returned.

3.1.3 Mock Providers

To provide the aforementioned mock data when it is requested, mock providers take a certain kind of request and return mock data conforming to the type of data expected by

the requesting application. This is the most flexible portion of Metis, and is meant to be easy to override with third-party providers. Currently, application developers do not check [28] if the data they are requesting appears to be valid. Thus, the mock providers currently in use are simplistic in nature, returning static or otherwise easily generated data. In the future, if the need arose for more realistic data to be returned, a new mock provider could be written and distributed to anyone using Metis without the need for any internal changes to the Metis system itself.

Coupled with the idea that mock providers should be modular, they must also be trusted. However, beyond this they do not require any special permissions, as they communicate only with the request router. As with the classifier above, we will discuss the specific feature of Android that satisfies the trusted aspect of mock providers in the next section. In systems where this or a comparable feature is not present, Metis would simply rely on the user to verify third-party mock providers. Other modifications are possible, but are outside of the scope of this thesis.

4 Implementation

Metis is implemented as part of the Android framework, as well as standard applications. The app classifier and mock providers are implemented as standard applications while the request routing is implemented as part of the Android framework itself. This approach was taken in order to make the user interface and mock providers easy to upgrade. This decision also allows us to introduce as little code as possible to the Android framework itself. By reducing the amount of code that runs with system privileges, the possible number of programming bugs and potential security holes is reduced.

4.1 Android Applications

Both the app classifier and the various mock providers are implemented as Android applications. We discuss each one in detail, but first examine the trust model which they both share. As mentioned in Section 3, both application components must be trusted by the user and the system. If either was not trusted, there is no way to guarantee that the

user's data is actually being protected. For example, if the user replaced the app classifier with untrusted code, there would be no way to verify that the untrusted classifier actually marked applications the user specified as untrusted. This classifier could easily mark every application, regardless of user preference, as trusted. Similarly, if a mock provider was untrusted, it could simply return the user's actual data in place of the mock data it should return.

To ensure that these scenarios are not possible, Metis registers two new permissions with Android, `SET_METIS_PREFRENCES` and `PROVIDE_MOCK_DATA`. Both permissions are given the `signatureOrSystem` protection level. This allows only system-level applications like the request router or applications which are signed with the same developer certificate as the defining application to request and be granted these permissions [5]. Therefore, only verified and system applications will be able to request these permissions, which are required to interact with Metis.

Specifically, `SET_METIS_PREFRENCES` is required by the app classifier to update the datastore with what permissions are considered dangerous and what applications are untrusted. It is worth noting that even querying the Metis datastore for which permissions and applications are currently considered dangerous and untrusted requires this permission. Such security is required to make sure that other applications cannot detect when a user has elected to give them mock data. In the same vein, `PROVIDE_MOCK_DATA` is required by any mock provider, and is required to both receive requests from the router and to return replies to those requests. Again, this is to prevent applications from listening for requests for mock data and inferring if the data they request is real or mocked.

4.1.1 App Classifier

The app classifier is responsible for managing and displaying the user interface along with setting and displaying the current state of the Metis datastore. The classifier listens for a broadcast from the Android system in which a new application has been installed, and then launches a modal dialog, asking if the newly installed application is trusted or untrusted. By presenting the user with the classification task during the install procedure, the user should be primed to reason about the newly installed application. Also, by using a modal,

we visually convey that this question is part of the installation procedure. For non-technical users, or those users who find the default privacy settings adequate, this will be the only interaction they have with Metis.

If the user is interested in changing the trustworthiness of an application or changing which permissions are considered dangerous, the app classifier can be launched like any other Android application. Since it is the only user-facing portion of Metis, the application itself is titled Metis. Upon launching, the application displays a simple menu, allowing users to manage their applications, or manage their privacy policy. Both items function similarly, displaying a list of trusted applications and dangerous permissions, respectively. We chose to display these lists, as opposed to their complements, because we expect that most applications will be considered untrusted by users, and most permissions are not dangerous. At the end of the list, an *add* button allows the user to add any of the currently untrusted applications or non-dangerous permissions to the trusted application or dangerous permission list. Users are also able to remove the current items from these lists. All preference changes are sent to the datastore in a background thread.

4.2 Android Framework Modifications

The request router is implemented by directly modifying the Android framework. In order to guarantee that every request for user data would be captured by the router, system-level modifications are the only choice. If a user-level application were used, applications which used direct communication with the intended data provider would be impossible to intercept. However, running as part of the Android framework does provide the router with better performance than otherwise possible. This speedup is mainly due to eliminating a context switch per message sent or received.

To support the design of a centralized router, we modified many of the existing Android classes related to content. Where requests for data could possibly end up, we added a hook which calls the `checkAccess` method of the router. In many places like the `LocationManager` and `TelephonyManager`, these hooks are simple additions. For other resources that do not pass through a high-level manager class like the Camera, we instrumented the `open` calls instead. When a request is sent to one of these original recipients,

it is then forwarded along with where it came from. The router uses this information to determine which permission the requesting application needed to be allowed to make this request. It then checks the Metis datastore to determine if this permission is considered dangerous. If it is not, the router will simply get the requested data and return it to the original recipient. If the permission is dangerous, another check is made to determine if the requesting application is trusted. Like before, if it is a trusted application, the router completes the request normally. However, in the case of an untrusted application making a request, the router will send the request to the mock provider registered to handle the specific kind of request being made.

4.2.1 Datastore

Besides its main duty of routing data requests, the router also handles access to the datastore. When users updates their preferences or a new mock provider is installed, the associated applications will send a request to modify some data in the datastore. The router checks for the proper permissions as discussed earlier, and then makes the actual changes. Since we do not store any structured data beyond a set, the datastore is implemented using Android's `SharedPreferences`. This allows for efficient random access which is well suited for the periodic lookups that routing requests generate.

4.3 Mock Data

To demonstrate Metis at work, we supply three mock providers which work with untrusted applications to provide mock locations, device ID's, and user contacts. While each of these mock providers is relatively simplistic, they can be replaced with more realistic providers if there is a need. Furthermore, mock providers for other kinds of data can easily be incorporated. Every mock provider registers at install time with Metis. During registration, the mock provider tells Metis what kind of requests it can handle, which is then saved to the datastore. If a new mock provider attempts to register for a request type that is already bound to another mock provider, the old binding is overwritten with the new mock provider.

4.3.1 Location

The location mock provider is a simple mock provider which registers for location data via the `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` permissions. When called, the mock provider will generate a random latitude and longitude roughly bounded to be within the continental United States. This initial location will be saved for a random duration between 30 minutes and 24 hours. If subsequent calls are made, a small random delta will be added to the initial location, which will then be updated with this new location. When the time from initial location generation is elapsed, all current location data will be cleared, and the next call to the mock provider will act as though it is the first call. This implementation generates mildly realistic location data which is completely different from a user's actual location.

4.3.2 Unique Device Identifier

Providing a mock identifier is very simple, and requires little code. This mock provider registers for requests against the device identifier via the permission `READ_PHONE_STATE`. We simply apply the SHA-1 hash function to the full package name of the requesting application. This allows for multiple calls from the same application to return a consistent value while thwarting cross-application tracking which using the device identifier as an invariant. This method is extremely simple, and does cause any two phones using Metis to report the same device identifier to the same applications. For now, this is of little concern as no known applications attempt to verify such information. If this approach were to become less useful, a simple update to the mock provider with a more sophisticated algorithm would solve the problem, and demonstrate the strength of the Metis model.

4.3.3 Contacts

The contacts mock provider is more involved than the previous two described due to the way Android represents a contact. It registers for all requests dealing with contacts, via the `READ_CONTACTS` and `WRITE_CONTACTS` permissions. When called, our mock provider returns a static list of three mock contacts. Any attempts to add to or edit these contacts are

allowed, but never persisted. This gives the illusion of being able to dynamically modify a user's contacts, but all attempts are simply discarded.

5 Evaluation

To evaluate the effectiveness of Metis, we wanted to ensure that Metis had acceptable performance to be usable and to explore how running Metis impacted many of the most popular free applications from the Google Play store [10]. All testing was done using a modified version of the Android 4.1.2 r1 "Jelly Bean" release.

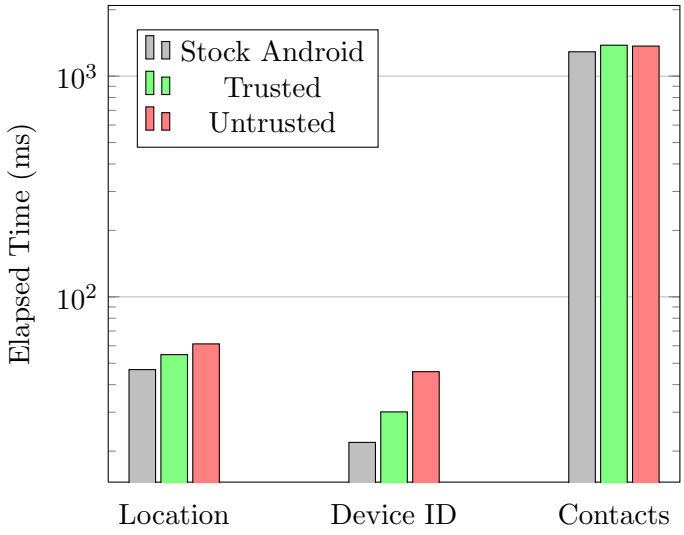

Figure 2: Average runtime for micro-benchmarks ($n = 10$)

5.1 Performance

To measure the performance of Metis, we created three micro-benchmarks which exercised our three mock providers by requesting the data they provided. We ran these benchmarks 10 times each on a stock Android system and on a system with Metis installed. On the Metis system we marked our application as both trusted and untrusted, and ran the benchmarks under both configurations. We also changed the permissions that our mock providers service to not dangerous, but the results were indistinguishable from the trusted application runs and thus omitted.

As Figure 2 shows, Metis introduces a modest overhead when retrieving both the location and device identifiers. We can attribute some of this overhead to the hook Metis makes into the location and telephony managers as discussed in the previous section. When an application is untrusted, we see an increase in overhead again due to the extra time required to call out to the mock providers. In the location benchmark, the increase from trusted to untrusted is minimal, since the mock provider itself executes just as quickly if not quicker than actually querying a real location sensor like the GPS. In the device identifier benchmark, the extra overhead is much more noticeable, because the real provider of the device identifier is a simple access whereas the mock provider computes a hash for every request. Unlike the other two, the contacts benchmark is quicker when tested with an untrusted application than when tested with a trusted one. This is most likely because of the static nature of our contacts mock provider.

Overall, we see that Metis incurs only a modest overhead in most scenarios. It is a bit concerning that performance is so closely tied to the relative efficiency of the mock provider when compared to the real provider. However, since all mock providers must be verified anyway, performance can be tuned to within an acceptable threshold before a new mock provider is accepted.

5.2 Usability

To ensure that Metis was not only performant, but did not interfere with the users' ability to use their applications, we downloaded the top 25 free applications from the Google Play store. Free applications are much more likely to be untrusted, since most users won't pay for applications they do not trust. Also, the prevalence of advertising libraries is much greater in free applications.

We ran two tests, the first with all applications set as untrusted and the permissions associated with location, device identifiers, and contacts considered dangerous. We then attempted to use the applications, and noted any usability issues. After this, we marked the permission allowing an application to utilize our coarse location as not dangerous, and made any application whose main purpose directly related to one of the aforementioned permissions as trusted. For example, if users downloaded the Skype application, presumably

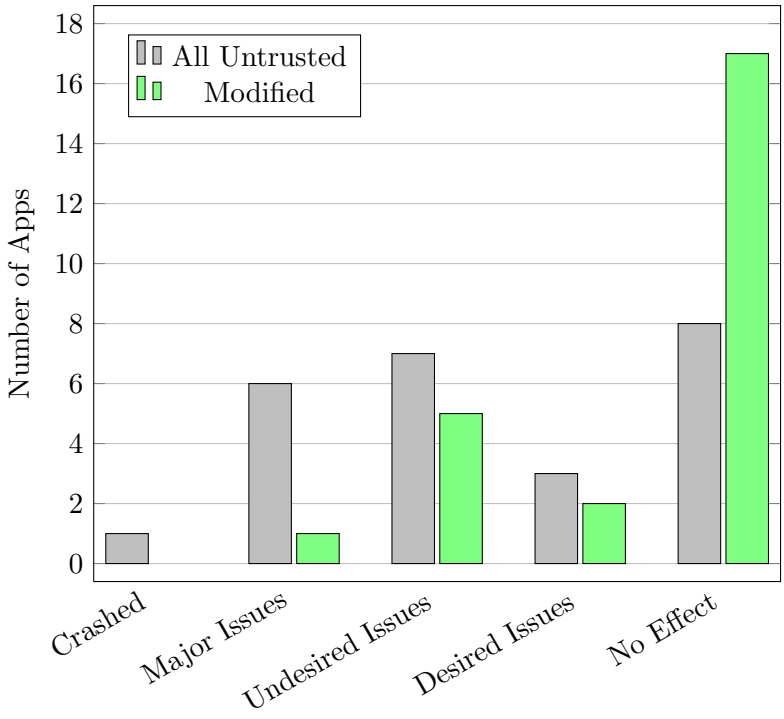

Figure 3: Application Usability Under Metis

they want to talk to some of their contacts via Skype. Trying to achieve this with mock contact data will fail for obvious reasons.

5.2.1 App Usability with Mock Data

We classified how the applications were affected by the mock data into one of five categories. Applications which failed to function when used with mock data were placed in the "Crashed" category. Applications which had their major feature or functionality impeded had "Major Issues". If a relatively minor, but still desired feature was impeded the application was placed into the "Undesired Issues" category. For some applications, Metis blocked a portion of the application which was not a directly listed feature or related to the advertisements. Those were placed in the "Desired Issues" category, and applications which Metis had no noticeable impact on were classified under the "No Effect" category.

From the results in Figure 3, we can see that even in the strictest setting, 44% of untrusted applications function correctly. Most of those that do not are applications like Skype in the example above. Once we correct for this, over 75% of applications function

properly, with only 4% of applications being unusable. These numbers show that Metis is extremely usable, even for security conscious users. More importantly, we observe that Metis fulfills its purpose of protecting user data privacy well, blocking advertiser's access to private data as intended.

6 Conclusion

It is a challenging task to protect user data privacy while minimizing the degradation in usability in smartphones. In this thesis, we proposed a solution called Metis to tackle this problem.

Metis is a system of loosely-coupled components for protecting a user's data privacy from unwanted access. It accomplishes this goal by supplying mock data to untrusted applications which request private data. We implemented Metis as part of the Android framework and standard applications. Then, we evaluated the performance of Metis and its usability. Our experimental results show that Metis is able to effectively protect user data privacy without degrading the usability, and the performance overhead incurred by Metis is modest.

6.1 Future Work

There are many avenues for further study and work relating to the Metis model for data privacy. While the user interface presented by the app classifier is clean and intuitive, more direct study on improving it could be beneficial. Usability is a key feature of Metis which can always be improved. In this vein, automatically classifying applications could help non-technical users have a more usable experience while not sacrificing data privacy. The automatic classification could be based on voluntary feedback from other Metis users as to how they have classified applications on their mobile systems. A central server could aggregate this data, and then make suggestions for new users based on the most popular trends.

Barring a vulnerability in the Android framework, Metis should not suffer from security issues directly. However, side-channel leaks such as a timing attack could allow an applica-

tion to detect when the data returned to them is from a mock provider. It would be best if the request router were able to make sure that requests sent to real and mock providers take a roughly equal amount of time. There is an inherent performance trade-off for this kind of protection, but further study into the value of such a trade-off is warranted. It would also be preferable to ship Metis with a more complete set of mock providers as well as more realistic ones. However, this is a minor issue due to the modularity of mock providers.

Performance can always be improved as well. Metis currently follows a conceptually simple model, but this leads to some inefficiencies in the implementation. There are hooks placed into some of the Android system code which could probably be optimized by doing the necessary checks and calls to the data providers at the source, instead of using the router as a proxy. Cutting down on inter-process communication like this would reduce the latency shown in Figure 2.

7 References

[1] (2011, May 10). *Twitpic Terms of Service* [Online]. Available: `http://twitpic.com/terms.do`

[2] (2013, March 2). *Adblock Plus* [Online]. Available: `http://adblockplus.org/`

[3] (2013, March 5). *SpiderOak* [Online]. Available: `https://spideroak.com/`

[4] 18 U.S.C §2511 (3) (2013).

[5] Android Developer Reference, `<permission>`. (2013, April 11) [Online]. Available: `http://developer.android.com/guide/topics/manifest/permission-element.html#plevel`

[6] Android Developer Reference, `android.test.mock`. (2013, April 3) [Online]. Available: `http://developer.android.com/reference/android/test/mock/package-summary.html`

[7] A. Karlson et al., "Can I Borrow Your Phone? Understanding Concerns When Sharing Mobile Phones," in *Proc. SIGCHI Conf. Human Factors Computing Systems*, Boston, MA, 2009.

[8] A.P. Felt et al., "Android Permissions Demystified," in *Proc. 18th ACM Conf. Computer and Communications Security*, Chicago, IL, 2011.

[9] E. Hayashi et al., "Goldilocks and the Two Mobile Devices: Going Beyond All-Or-Nothing Access to a Device's Applications," in *Proc. 8th Symp. Usable Privacy and Security*, Washington, D.C., 2012.

[10] Google Play Store. (2013, April 13) [Online]. Available: `https://play.google.com/store/apps/collection/topselling_free`

[11] H. Haddadi et al., "Targeted Advertising on the Handset: Privacy and Security Challenges," in *Pervasive Advertising*, 1st ed. London, UK: Springer, 2011, ch. 3, pp. 119-137.

[12] I. Leontiadis et al., "Don't kill my ads! Balancing Privacy in an Ad-Supported Mobile Application Market," in *Proc. 12th Workshop on Mobile Computing Systems and Applications*, San Diego, CA, 2012.

[13] J. Brodkin. (2012, July 31). *Dropbox confirms it got hacked* [Online]. Available: `http://arstechnica.com/security/2012/07/dropbox-confirms-it-got-hacked-will-offer-two-factor-authentication`

[14] J. Mello Jr. (2012, May 18). *Facebook Hit with Lawsuit Alleging Privacy Wrongs* [Online]. Available: `http://www.pcworld.com/article/255803/facebook_hit_with_lawsuit_alleging_privacy_wrongs_seeks_10k_for_each_member.html`

[15] J. Seifert et al., "TreasurePhone: Context-Sensitive User Data Protection on Mobile Phones," in *Proc. 8th Int. Conf. Pervasive Computing*, Helsinki, Finland, 2010.

[16] M. Meeker, "Internet Trends Report," Kleiner Perkins, New York, NY, 2012: `http://www.kpcb.com/file/kpcb-internet-trends-2012`

[17] N. McAllister. (2013, February 2). *Twitter breach leaks emails, passwords of 250,000 users* [Online]. Available: `http://www.theregister.co.uk/2013/02/02/twitter_breach_leaks_user_data/`

[18] N. Vratonjic et al., "Ad-blocking Games: Monetizing Online Content Under the Threat of Ad Avoidance," in *11th Annu. Workshop Economics of Information Security*, Berlin, Germany, 2012.

[19] O. Riva et al., "Progressive authentication: deciding when to authenticate on mobile phones," in *21st USENIX Security Symp.*, Bellevue, WA, 2012.

[20] Providing Mock Location Data. (2013, April 7) [Online]. Available: `http://developer.android.com/guide/topics/location/strategies.html#MockData`

[21] P. Hornyack et al., "These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications," in *Proc. 18th ACM Conf. on Computer and Communications Security*, Chicago, IL, 2011.

[22] P. Pearce et al., "AdDroid: Privilege Separation for Applications and Advertisers in Android," in *7th ACM Symp. Information, Computer and Communications Security*, Seoul, Korea, 2012.

[23] R. Singel. (2011, June 20). *Dropbox Left User Accounts Unlocked for 4 Hours Sunday* [Online] Available: `http://www.wired.com/threatlevel/2011/06/dropbox/`

[24] S. Shekhar et al., "AdSplit: Separating smartphone advertising from applications," in *21st USENIX Security Symp.*, Bellevue, WA, 2012.

[25] V. Silveira. (2012, June 6). *An Update on LinkedIn Member Passwords Compromised* [Online]. Available: `http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised/`

[26] W. Enck et al., "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *9th USENIX Symp. Operation Systems Design and Implementation*, Vancouver, B.C., 2010.

[27] X. Wei et al., "Permission Evolution in the Android Ecosystem," in *Proc. 28th Annu. Computer Security Applications Conf.*, Buena Vista, FL, 2012.

[28] Y. Zhou et al., "Taming Information-Stealing Smartphone Applications (on Android)," in *4th Int. Conf. Trust and Trustworthy Computing*, Pittsburgh, PA, 2011.