# MyShield: Protecting Mobile Device Data via Security Circles

Rodney Beede, Donald Warbritton, and Richard Han
Department of Computer Science, University of Colorado at Boulder
Contact{rodney.beede or donald.warbritton}@colorado.edu

## Abstract

As smartphones and the third party applications they run become ever more popular, the amount of personal information at risk for users continues to increase. To protect the privacy of mobile user data such as a user's location, contacts, phone number, etc., mobile applications on the Android and iOS platforms typically request the user who is installing their application to grant them permission to a variety of data on the phone that they want to run the application. The user either approves or rejects the request - there is no middle ground. Furthermore, there is no guarantee that the app will respect the privacy of the user by accessing a minimum amount of information or ensuring that data does not leave the device unnecessarily. To remedy this situation, we introduce MyShield, a system that protects user data from mobile applications by supplying anonymized data when desired by the user, and in a way that is transparent to applications. MyShield incorporates our concept of Security Circles - an intuitive way to allow users to control what applications get access to which data by grouping applications together with similar trust levels. The applications in a given Security Circle are then subject to the same anonymization policy. We present an implementation of MyShield and Security Circles in the Android operating system.

## General Terms

Mobile Privacy, Anonymization, Security Circles

## 1 Introduction and Motivation

The advent of widespread smart phone usage has lead to a new communication revolution. Millions of these devices are in use today and provide many advanced capabilities to their users. Modern smart phones have large market places with up to hundreds of thousands of applications (known as apps) that provide a variety of functionality. These apps range from games to cameras to maps and route guidance.



One important consequence of this technological and social revolution has been the impact on the privacy of the mobile user. Smartphones contain an ever-increasing amount of personal data. This information is obtained from a variety of sources - input from the users, readings from various sensors (GPS, accelerometer, etc.), data deposited by third party applications, as well as video and image data captured from the camera. Mobile devices also contain unique identifying information, such as the IMEI or MEID [3] [1], that could be used to identify and track a user.

The current framework for managing privacy policies on mobile devices is a take-it-or-leave-it model that leaves significant room for improvement. Consider the process for installing an app from an app store. On both the iOS [2] and Android [10] platforms, the user is presented with a list of permissions that an application requires when it is being installed. If the user desires to grant all the requested permissions, the app may be installed. If not, it is not possible to install the application. This take-it-or-leave-it model allows an application developer to request permission for information for which the application has no legitimate use for the end user. The user is faced with the stark choice of either granting all such permissions, or not installing the application at all. Alternatively, the application may request more detailed information than the user is comfortable revealing; for instance: an application that suggests restaurants may realistically only need to know where the user is to within a few city blocks, but actually requests GPS access that reveals the user's location to within a few meters. Allowing users to control the degree to which a given data type is revealed is not a type of granular data management that is available in current smartphone OS iterations.

In order to address the issues discussed above, we present MyShield: a system designed for allowing simple, intuitive, and granular control over the data an application may access. As a design requirement for our system, we did not want to have apps become inoperable due to runtime exceptions. In other words, everything should behave as before, with the exception that privacy is maintained. We accomplish this goal by allowing the user to specify how much trust they have in a particular application. Our chosen metaphor is that applications should be placed into a particular circle of trust, similar to the way that people develop social circles with varying levels of trust (and inspired by the circle mechanism used in Google+ [11]). For instance: one's family circle is likely to

be highly trusted, while the circle of people they work with may be somewhat less trusted. In the same way, applications that are placed into a trusted circle are allowed complete access to the information for which they've requested permission, while applications in an untrusted circle may be given false information which does not compromise the users's privacy. Applications that do not precisely fit either of these descriptions may be somewhat trusted - data supplied to them will be based on real data, but anonymized in order to maintain some degree of privacy for the user. This paradigm will become ever more important as the number of malicious applications continues to rise [7].

Figure 1 shows the interworking between our MyShield app and the shim layer responsible for implementing the MyShield privacy policies.

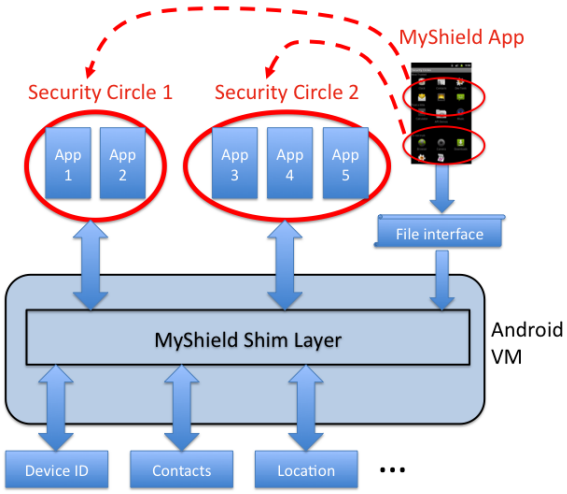



Figure 1: A user can group mobile applications into different security circles via the MyShield management application. The MyShield layer will then shield user data like device ID, contact information, and location from applications using a unique anonymization policy for each security circle.

Our system enables the user to easily control their privacy by limiting what applications can actually read data while also still allowing for those applications to run. This changes the existing model from install with whatever access permission the app requests or dont install at all to a model of user configured trust with control over personal information.

The contributions of the paper are as follows:

- The implementation of a shim layer in the Android operating system capable of controlling access to to sensitive information.
- The implementation of a system for managing the amount of data applications are capable or accessing.
- The proposal and implementation of strategies for anonymizing user data.
- The proposal of a system allowing for the extension of our current solution without having to make any modifications to the actual system.

We next describe related work in Section 2 and our design goals in Section 3. We introduce the concept of security circles in Section 4, and our system architecture in Section 5. We then describe the anonymization strategies for device ID, location, and contact information in Section 6. Our implementation of these strategies is discussed in Section 7, and an evaluation is presented in Section 8. We describe future directions in Section 9 and conclude in Section 10.

## 2 Related Work

The state of permission management on Android can be concerning for anyone. Users want the ability install third-party applications, but being sure that user's trust in that application is not being abused is difficult. The work done by Enck et al. [8] showed that the lack of transparency present in Android allowed many popular apps to misbehave when trusted with potentially sensitive user information. This should be cause for concern among users - applications may be actively exploiting a lack of granular permission control for less than wholesome purposes on a regular basis.

In some cases, even data that seems quite innocuous may be used to discern the behavior of the user. The authors of [16] were able to classify some of the actions of users carrying mobile devices using only accelerometer data. In fact it is the case that access to accelerometer data for Android applications requires no permission at all. These results indicate that data protection may be important even across data sources that seem to be of little consequence.

In response to the lack of granular permission management present on Android, there have been at least two other attempts at offering a more comprehensive system. While these systems are powerful in some ways, we believe our solution provides capabilities offered by neither while simultaneously avoiding some of the disadvantages they inherently contain.

The Android application LBE Privacy Guard [14], available in the Android market for free, is capable of a high degree of permission management, along with a built in firewall. Privacy Guard intercepts sensitive API calls, and allows the user to decide whether or not they want the application making the call access to the data it is requesting. Granting permission to the application allows it to continue as normal, while denying permission results in blank data being returned. While this application is very powerful, we believe there are two ways in which our system can improve upon it. Firstly, it is only possible to run this application on phones where the user has root access. To the best of our knowledge, this is not the case with **any** popular phone out of the box, and the process of acquiring root access is often quite daunting, warranty voiding, and not standardized. Furthermore, even users with root access to their device may not be comfortable installing an application with unrestricted permission. Secondly, decisions about application permission are strictly binary - an application must either be completely trusted with access to some data, or not at all. We believe there should be room for a "somewhat trusted" metaphor.

The second attempt at a solution comes via the popular custom Android distribution CyanogenMod [13]. Recent

versions of CyanogenMod include logic for toggling specific permissions per app using the same system Android uses internally. While this allows the granular control we believe is necessary, we don't believe this solution is sufficient. In CyanogenMod, if an application makes a request for data for which it does not have permission, the operating system throws a SecurityException. Because it is not possible to install an application from the Android market without granting every permission the application requests, it is very likely that the application is not designed to catch SecurityExceptions. The result of an uncaught SecurityException is a crashed application, potentially disabling the functionality of the app completely. This type of behavior is likely to result in allowing the same set of permissions that the application originally requested, thereby invalidating the approach.

In order to allow more granular permission management, we have elected to implement anonymization techniques capable of allowing some degree of information through. In areas such as location anonymization, where significant work has already been done [12] [15] [18] [17], we have not attempted to present groundbreaking new work. Rather we have implemented simple methods with the idea that an extensible system will allow for other techniques to be inserted in the future.

## 3 Design Goals

There are several design goals that we believe must be achieved in order to advertise our system as a significant step forward over existing attempts. Further, we believe that a system that meets these goals will be much more attractive to mobile carriers wishing to promote greater security to their customers.

Our first concern was that any realistic system should be completely transparent to applications installed by the user. Previous work whose solution is to throw SecurityExceptions when applications expect to access data create an environment where users are much more likely to either give all permissions to every application (solution invalidated), or abandon the system entirely. Our system should cause as little frustration as possible, and should be completely invisible much of the time.

This introduces our second design goal - the system should be seamlessly integrated at a low level. As an intrinsic piece of the operating system, the largest degree of power and configurability is possible. This also helps to ensure that applications are unable to interfere with the operation of the system.

The third design goal is that the system should be intuitive to the user. We believe that one of the primary problems with the current permission structure is that it fosters confusion: users are presented with a (potentially) long list of permissions the application requires, but little information about whether the user should be concerned about their privacy is given. Our system should remove the user to an abstraction where they must only decide what level of trust they have in the application (figure 2). Our solution to this is discussed in section 4.

Our fourth design goal is that there should be some in-between for application permission - it does not always make sense to make a binary decision between allowing or denying data access. Our system should be able to provide some level of anonymization of user data such that some applications are able to continue functioning while not disclosing more personal data than necessary. There is a large and varied amount of data that could be protected on a smart phone, and as such techniques for anonymization of this data are similarly varied. We discuss our strategies in section 6.

Our fifth design goal was that the system should not require root level permission to run. Users may be uncomfortable installing an application with root access to their device. If possible, the system should be able to operate based only on modifications to the operating system, and standard application level access where user control is desired.

The sixth and final design goal is that the system should be extensible. As already mentioned, data anonymization is a non-trivial field, and we do not claim to have solved the problems necessary for a perfect system. For this reason, we believe that our system should be able to be extended by others without having to make changes to the system itself. A method for allowing users to easily drop in new anonymization techniques would ensure that the system continues to grow and remain useful into the future.

## 4 Security Circles

One of the difficulties in ensuring that users are aware of the pitfalls of granting a slew of permissions to a particular application is that there is no indication of the threat posed by the application when their personal information is considered. In fact, the behavior of the Android Market is such that, if an application requests a long list of permissions, not all of the requested permissions are shown to the user at install time. It is only if the user elects to access a different view that the entire list is presented. The user is then forced to decide if the utility of the application is worth the risk of the disclosure of potentially private data.

### 4.1 Motivation

While, from a technical perspective, there is nothing wrong with this model, we believe that a more intuitive interface is possible. An ideal solution would keep the user from being overly concerned with what access the application is allowed - some other abstraction should be responsible for that. To this end we propose a new paradigm we call Security Circles. Rather than asking which permissions should be allowed or denied for a particular application, the user should only decide how much they trust the application. Much in the same way that humans develop social circles with varying levels of trust, we may also develop circles of trust for applications. By assigning applications to circles defining different levels of trust, users implicitly grant corresponding levels of permission. Popular apps from reputable developers are likely to be very trusted, as well as applications with fundamental requirements for access to sensitive information, such as route guidance systems. On the other hand, wallpaper or game apps likely need to be trusted much less in order to run. This paradigm fulfills both the second and third design goals - it presents an intuitive interface for permission management, and it achieves granularity. By al-

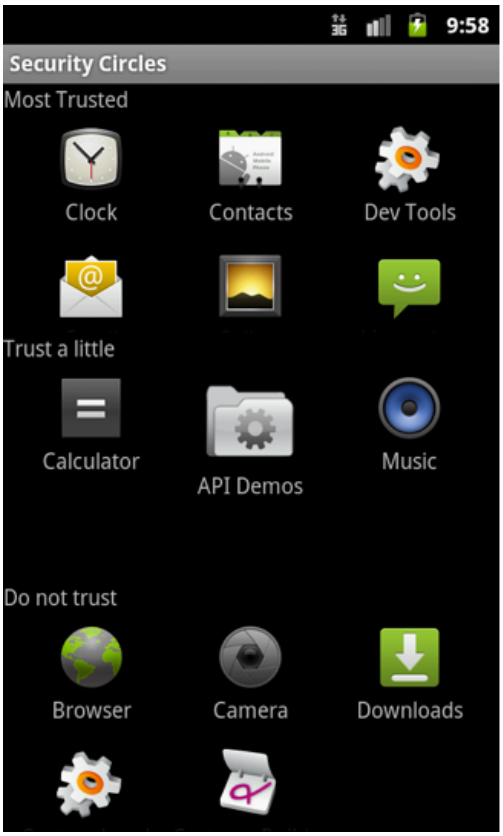



Figure 2: MyShield management application implementing Security Circles, with a few applications in each circle

lowing for several different circles, say: trusted, somewhat trusted, and untrusted, there is a logical progression from one level to the next that avoids the binary yes or no decision imposed by other solutions. Furthermore it creates an architecture for more circles, and in fact customization of circles, in the future.

### 4.2 Definition

At the most concrete level, a Security Circle is a list of rules concerning what transformations should be applied to which data. When an application belongs to a particular circle, every request made through sensitive APIs will be subject to the regulations of that circle. An example: say a mapping application is placed in a circle where access to location information is not allowed. Although this application will still be able to register for and receive location updates, the information contained in the updates will not be genuine (a discussion of what might be returned is covered in section 6.2). Rules for every type of data may be defined for each circle, allowing a complete privacy policy with little effort required by the user. In MyShield, we have limited ourselves to a reasonable three circles - Trusted, Trust a Little, and Untrusted. The three circles that we have implemented are shown in Figure 2, where each circle contains its own set of applications. The methods for assigning applications to circles and the consequences of belonging to different circles will be discussed in greater detail in sections 5.2 and 6 respectively.

## 5 System Architecture

In order to seamlessly integrate our system at a low level, we chose to modify code from the Android Open Source Project [9]. Android is an open source operating system based on a linux kernel that is designed for mobile devices. Not only is Android the most popular smartphone OS on the planet [5], but it is designed so that anyone can download and build the source code with relative ease. This means the developing modifications to a popular production system can be done in a reasonably short amount of time. We chose to develop on the 2.3.7b1 branch of Android, as it was the most recent and widely used branch for which source code was available (since the time we started development, the source code for 4.0 has become available, although we have not investigated what changes may be necessary to port our system).

### 5.1 Shim Layer

After reviewing where we felt previous work in this area had both succeeded and failed, along with the design goals we felt had to be met in order to be successful, we decided that a combination of the two solutions discussed previously would be ideal. We wanted to incorporate both the ability to send altered data back to applications while avoiding the requirement that a third party application operate with elevated privileges. The only realistic method for meeting this second goal is that we make modifications to the Android operating system itself. By creating what amounts to a shim layer that exists between applications and private data, we can monitor what applications are making requests for what data. This shim layer allows a great deal of control over how requests are handled. It also allows for the desired transparent operation - applications are not privy to how the operating system handles their requests, so they are unable to know if the result of their request has been tampered with along the way.

The use of a shim layer also allows for the sixth design goal to be met - extensibility. If different techniques for data tampering are desired in the future, it is only necessary that these techniques conform to some interface, and that the shim layer knows of their location. The shim layer is shown in Figure 3, inserted between the applications and the underlying Android VM. In reality, its implementation is spread throughout the Android source code, intercepting calls to sensitive data from applications and replacing the results with suitably anonymized data as defined by each circle's anonymization policy.

### 5.2 MyShield Management Application

As our proposed system exists at a low level within Android, there is no direct method for configuration of which apps belong to which circles. In fact, the shim layer has no knowledge of applications, only the user IDs that the apps are associated with. To act as an interface to the shim layer for the user, we created the simple MyShield App (figure 2). This app is responsible for keeping track of what Security Circle the applications present on the system belong to, as well as moving apps between circles. The MyShield app presents an intuitive interface for users to manage their privacy preferences. The window is divided into three parts,

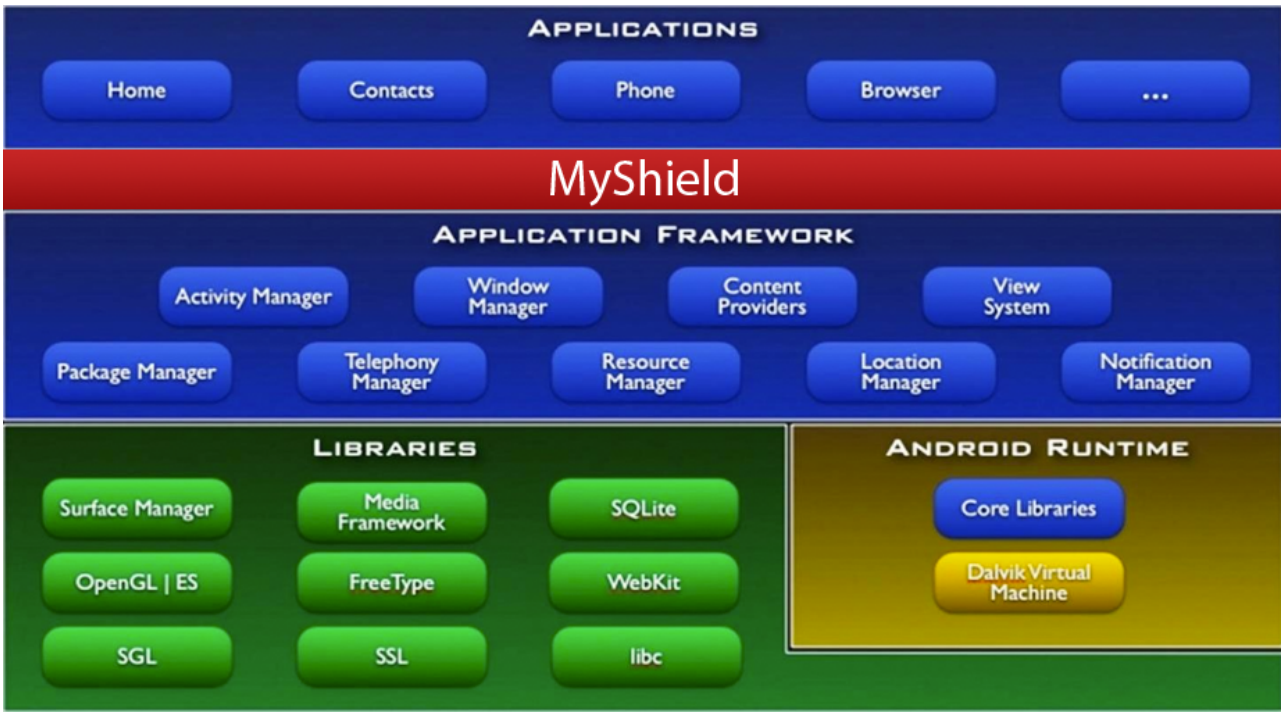



Figure 3: Location of Shim Layer in Android

where each part represents a particular security circle. All applications start in the most trusted circle - in order to move an app to another circle, the user must only locate that app's icon and drag it to another portion of the screen. We believe this nicely accomplishes our third design goal for intuitive system configuration.

The interface between the MyShield app and the shim layer is a configuration file for which only the MyShield app has write permission (although the file is universally readable). Each call to sensitive data that is intercepted by the shim layer causes MyShield to check the configuration file for the ID of the calling application to find the appropriate security circle and anonymization policy for that data call. This allows a unidirectional flow of information straight from the user into the system. Changes to the circle of an application are reflected by new anonymization in the very next call to sensitive data.

## 6 Anonymization Strategies

While controlling access to private information is useful, that usefulness can be inversely proportional to how quickly applications that rely on that data become handicapped or completely disabled. For instance: an application designed for recommending nearby restaurants is likely to continue working quite well if the user's location is accurate to within a couple of city blocks. On the other hand, that same application may become unusable if the location data is wildly inaccurate or completely unavailable. For this reason it is important that we make intelligent decisions about how moving an application from one Security Circle to another will affect its behavior; furthermore, complicated explanations about how the system will behave are undesirable for end users - the system should "just work".

Listing 1: Anonymizing the device ID according to the Security Circle

```
String circleLevel = getCircle(uid);
String deviceId = mPhone.getDeviceId();

if("TRUSTED".equals(circleLevel))
  return deviceId;
else if("TRUST_A_LITTLE".equals(circleLevel))
{
  String savedId = getSavedId(uid);
  if (savedId != null)
    return savedId;
  return generateRandomId();
}
else if("UNTRUSTED".equals(circleLevel))
  return generateRandomId();
else
  return deviceId;
```

### 6.1 Anonymizing Device ID

In considering how unique device identifiers should be treated, we developed two paradigms corresponding with the Trust a Little circle and the Untrusted circle respectively (listing 1). For applications in the Trust a Little circle, the device should be consistently identifiable to **that** application for the time the application is installed. We accomplish this by generating a random device ID for each application. The first time an app in the Trust a Little circle makes a request for a device ID, a random ID is generated and written to a configuration file. On that and all subsequent requests for

device ID from that application while it is still in the same circle, this randomly generated ID will be returned. In this way the application may track the device while it is installed, but the user has not had to give up any actual data. For applications consigned to the untrusted circle, we attempt to avoid supplying any useful information whatsoever. For this reason, we choose to generate a new random ID each time. Not only is the user's actual information protected, but it is not possible even to generate usage statistics using this data alone.

## 6.2 Anonymizing Location Data

Knowing the location of a user can be a powerful tool, a tool that many popular apps now take advantage of to provide a wide variety of services. However, accurate data about where someone is and has been could be considered very personal, and ensuring that only authorized and trusted parties have access to this information is extremely important. Despite the sensitivity of this information, many apps request access to it whose purpose is not related to location. We provide two strategies for anonymizing device location.

Listing 2: Function used to round latitude and longitude values to one decimal place

```
private double anonymizeNumber(double value){
  long foo = (long) (value * 10000);
  return ((double) foo) / 10000.0d;
}
```

In the case of the Trust a Little circle, we "fuzz" the device's true location (listing 2) in order to maintain some privacy while not disabling applications whose functionality is based on accurate location data. This is reasonably simple as latitude and longitude are represented as double precision floating point numbers on Android: the more decimal degrees are removed from the actual latitude and longitude coordinates, the less accurate the resultant location becomes. Our system currently anonymizes the location data for applications in the Trust a Little circle to what could be considered city level - we round up to the first decimal place, introducing an error of roughly 10 kilometers. This allows location based applications to continue working while disclosing relatively little information.

For application in the Untrusted circle, we neglect to provide any location data based in reality. Although one could imagine a variety of coordinates suitable for falsifying location, we choose (0,0) for latitude and longitude. We briefly considered providing random coordinates, but dismissed this idea as not being significantly more appropriate than (0,0), and there is potential for creating bizarre behavior in applications (instantaneous global transportation may not be a test case for every developer). Alternatively, coordinates located in the user's country could be used if some localization settings are determined by location.

## 6.3 Anonymizing Contacts

Not all sensitive information on a device is necessarily private to the device owner - the individuals whose personal information resides in the device's list of contacts are also exposed to privacy disclosures. Even if a user is typically lax in monitoring their own information, they may not feel comfortable exhibiting the same attitude towards their friend's privacy. Contact data consists of names, phone numbers, e-mail addresses, physical addresses, photos, and social network links. Because of this it may be desirable to only allow bits of the information typically contained in a single contact entry.

Due to the fact that contact data may be private to people besides the user, we believe a strict approach to allowing access should be followed. For applications in the Trust a Little circle, contact names could be reduced to the contact's first name and the initial of their last name (alternative approaches might be to return only initials, only return one name, etc.) . For other fields, such as phone number and email address, it makes little sense to attempt anonymization. If either of those fields is only partially correct, it is functionally useless. We believe that these fields, along with any others (photo, mailing address, etc.), should simply be returned as empty. This conservative approach protects the data of other people (potentially a large number of people), and enforces the idea that it is inappropriate for all but the most trusted applications to be able to access such a large and personal dataset.

Applications belonging to the Untrusted circle are simply disallowed from any information. As retrieving contact data is a database operation, the appropriate response in this case is to return zero rows.

## 6.4 Anonymizing Other Data Sources

Other private information often shared between the user and other parties are text messages. Strictly speaking, the API for accessing text messages stored in the user's inbox is not public; however, because the system is open, resourceful developers have determined the methods by which official Google applications access the the user's inbox. While not officially supported (and in fact explicitly discouraged [4]), there are many tutorials on the internet for accessing this data. A popular reason for accessing this data is for use in building predictive texting systems based on past trends. In this case it would not make sense to rearrange or otherwise mangle the content of the message. However, if only the data from the owner of the device is required (as in the previous example), incoming messages could be denied to requesting applications. For truly untrusted applications, returning no data is an obvious solution, although in some cases returning text from other sources could also be appropriate.

# 7 Implementation

Our system has two distinct components that work together to achieve our goals: an application installed on the device that allows the user to assign apps to the desired permission circle, and a shim layer in the Android operating system capable of intercepting requests for sensitive information. This builds on top of the permission management system already present in Android with more sophisticated controls. In this way we have avoided replacing any functionality in order to ensure that application and operating system behavior is modified as little as possible, and only in very specific ways.

## 7.1 Android OS

Inserting code to create the shim layer responsible for the actual implementation of our security circles design proved to be difficult in ways we did not expect. While the Android project is well documented from the point of view of an application developer, there is significantly less information available to those attempting to make fundamental changes to the operating system. Because of this, we spent a not insignificant amount of time simply becoming familiar with the inner workings of Android. Due to the large amount of information it may be desirable to enforce greater control over, we decided to prioritize which areas users would most likely want to see managed by our system. It was our decision to first focus on the control of location data and the device's unique identifier (IMEI for GSM devices, MEID for CDMA devices). Clearly pairing a unique ID with location data may be a large privacy disclosure.

This is not simply an imagined danger - this permission pairing is present in many popular apps: SoundHound, Facebook, Voxer, Angry Birds, Skype, Fruit Ninja Free - all top apps in the Android Market - require both location information and access to the device's phone number and serial number. In some cases, such as with Fruit Ninja Free, the premium version of the application requires neither of these permissions. This seems to strongly indicate that the extra permissions are being used for advertisement based on user information.

Because public documentation for the application framework is lacking, we had to devise other methods for educating ourselves. While not especially expedient, the most reliable method was to first examine the steps an application must take to obtain privileged data. Using the classes applications communicated with for this access as a starting point, we followed the chain of invocations, attempting to determine where our shim layer should attach. Unix utilities like find and grep became invaluable tools for examining such a large codebase. Unfortunately, it is not the case that writing one piece of the shim layer to manage a particular class (such as the LocationManagerService) is particularly helpful for writing other pieces of the shim layer. Methods for retrieving one type of data can be wildly different from others. Due to these complications, efforts to extend the shim layer across all components of the application framework are ongoing.

## 7.2 Security Circles Management Application

The GUI interface that users use to place applications into circles was written using the Java language using Android APIs. It is a regular Android application that runs without any special permissions or installation. The application/activities (an application can have multiple launchable activities in Android) icons and labels for the installed applications are retrieved using standard Android APIs and rendered onto a GridView in a similar fashion as the Android OS system application launcher works. The configuration of circles is stored in the GUI application's internal storage (under the /data/apps folder on most Android systems) as a world readable (not writeable) tab-delimited file. The first entry contains the package name (required so the GUI can load icons into the correct circle), the UID (used by the shim layer), and the chosen circle. The shim layer simply reads this text file each time it sees an application trying to access a protected sensor or data. No communication between the GUI application and shim layer is required. When the user updates an application's circle, the configuration file is rewritten.

## 7.3 Controlling Access to Location Data

The process for obtaining location data in an Android application can be broken down into two distinct steps: first the developer must create a listener class implementing the required methods for receiving location information (onLocationChanged(), onProviderDisabled(), onProviderEnabled(), and onStatusChanged()). Second, the application must make a request to the operating system to register the listener class for updated location information (returned by calling the listener's onLocationChanged() method). We theorized early on that the shim layer would have to be involved in the second step in order to affect the data the requesting application received.

Listing 3: Location data is given to location listeners

```
final int N = records.size();
for (int i=0; i<N; i++) {
  UpdateRecord r = records.get(i);
  Receiver receiver = r.mReceiver;
  boolean receiverDead = false;

  Location lastLoc = r.mLastFixBroadcast;
   if ((lastLoc == null)
     || shouldBroadcastSafe(location,
       lastLoc, r)) {
    if (lastLoc == null) {
     lastLoc = new Location(location);
     r.mLastFixBroadcast = lastLoc;
    } else {
     lastLoc.set(location);
    }
    if (!receiver.
      callLocationChangedLocked(location)) {
    ...
    }
   }
   ...
}
```

The class LocationManagerService is responsible for registering new listeners, maintaining a list of all registered listeners, and passing updated location information along to the correct subset of these listeners. The LocationManagerService code contains several other classes used for keeping track of listeners along with data such as what type of LocationProvider (e.g. GPSLocationProvider) the listener requested. The two classes the shim layer is concerned with are UpdateRecord and Receiver. An UpdateRecord represents a particular listener, while a Receiver represents a wrapper used for receiving location updates.

When a new listener is registered, a new UpdateRecord is created that contains the listener (the list of listeners mentioned is actually a list of UpdateRecords). Each UpdateRecord in turn contains a Receiver responsible for transferring the location data to the application containing the listener. Shown in listing 3 is the code snippet responsible for hand-

ing over new location data to the Receiver associated with a particular listener. This provides a key insight into the way the shim layer works for location data: if changes are made to the callLocationChangedLocked() method in the Receiver class, the location information eventually transferred to the listening application can be manipulated.

Listing 4: AnonyReceiver is a subclass of Receiver for anonymizing location data

```
private class AnonyReceiver extends Receiver{
  private Receiver originalReceiver;
  public AnonyReceiver(
      final Receiver originalReceiver) {
    super(originalReceiver.mListener);
    this.originalReceiver =
      originalReceiver;
  }
  @Override
  public boolean callLocationChangedLocked(
      Location location) {
    // Modify location
    Location l = new Location(location);
    l.setLatitude(anonymizeNumber(
      l.getLatitude()));
    l.setLongitude(anonymizeNumber(
      l.getLongitude()));
    l.setBearing(anonymizeNumber(
      l.getBearing()));
    l.setAltitude(anonymizeNumber(
      l.getAltitude()));
    l.setSpeed(anonymizeNumber(
      l.getSpeed()));
    return
      super.callLocationChangedLocked(l);
  }
}
```

We avoided making any fundamental changes to this system by creating new classes that extend the Receiver class. Shown in figure 4 is the entire implementation of the anonymizing receiver class designed for reducing the accuracy in the reported location (the anonymizeNumber() method is shown in listing 2). In doing this, we are able to override the callLocationChangedLocked() method. As shown in listings 4 and 5, we are able to make whatever modifications we desire to the location before it is passed on to the application. We believe this to be an elegant and easily extensible solution that allows for novel anonymization techniques to be applied little modification to the code already written.

Listing 5: Location modifications in the FakeReceiver class

```
@Override
public boolean callLocationChangedLocked(
  Location location) {
 // Modify location
 Location l = new Location(location);
 l.setLatitude(0.0d);
 l.setLongitude(0.0d);
 l.setBearing(0.0f);
 l.setAltitude(0.0d);
 l.setSpeed(0.0f);
 return super.callLocationChangedLocked(l);
}
```

As a simple example, this is sufficient to show that our shim layer is working correctly for location data, but it does not account for the Security Circles system. In order to integrate the two pieces, we cross reference the UID of the application requesting a listener with the Security Circle parameters - this determines what Receiver class will be associated with the UpdateRecord for the application (listing 6).

Listing 6: Receiver type is based on Security Circle

```
String circleLevel = getCircle(uid);
if("TRUSTED".equals(circleLevel))
   mReceiver = receiver;
else if("TRUST_A_LITTLE".equals(circleLevel))
   mReceiver = new AnonyReceiver(receiver);
else if("UNTRUSTED".equals(circleLevel))
   mReceiver = new FakeReceiver(receiver);
else
   mReceiver = receiver;
```

## 7.4 Controlling Access to Device ID

Devices connected to GSM or CDMA networks contain a serial number called the IMEI or MEID, respectively, that is used to identify the device on the network. Although this number is not necessarily directly tied to the subscriber, it is unlikely that a user would switch devices often enough to avoid being closely associated with it. Application developers may use this ID to track users for advertising, determining what users have purchased a premium version of the app, or for analytics data. While there may be legitimate uses for this data, a user may not be comfortable exposing this semi-permanent identifier to every application. In the same way that location data can be altered before it is returned according to the requesting application, we created the opportunity for adjusting the device's ID if a user so desires. In order to return altered information to the requesting application, we added the code in listing 1 to the PhoneSubInfo class.

Applications in the trusted Security Circle are privy to the actual device ID - no changes are made for applications in this circle. Decisions about what information should be returned for applications in the other two circles is a bit more tricky than for location data - there is no analogous method for reducing the accuracy of a unique identifier. Methods for varying levels of anonymity for this field are discussed in Section 6.1

## 7.5 Controlling Access to Contacts

Access to contact data is provided through a ContactProvider service on the device. The Android OS has a SQLite database that contains the information and is restricted based on the process uid to application permissions in the system. Android's provided API allows for an application to request read and/or write access to the contacts. The code for determining this access is found in the Android source at packages/providers/ContactsProvider/src/com/android/providers/contacts/ContactsDatabaseHelper.java in the hasAccessToRestrictedData() method. When an Android application wishes to access contact data, it calls the query method on a ContentResolver with the desired contact attributes (name, address, phone number, e-mail, photo). In

return a Cursor object of database results is returned that can be iterated over. A similar method is used to write contact data.

Because the hasAccessToRestrictedData() only returns a boolean code, modification to track which application is requesting the contact data and which circle they are in would have to be determined elsewhere. When the query method in the ContactsDatabaseHelper is called one would have to pull the current uid from the Binder instance to make the appropriate anonymization level (security circle) choice. An emulation of the database Cursor would have to be returned that returned the anonymized data instead of the underlying real data in the actual SQLite database.

## 8 Evaluation

Staying in the vein we have thus far explored, we chose to evaluate our system by using applications which rely on location and device ID information to operate. To satisfy our design requirements, the applications should first continue to behave as normal, and second only have access to the information allowed by the Security Circle to which they are assigned. For consistent results, we used an Android emulator running our custom build of Android 2.3. Location data was supplied to the emulator via a telnet connection - in this way we were able to supply exact coordinates time after time. The device ID for our emulator was always 000000000000000. The coordinates we used for location testing were (-105.2641318, 40.0082298), an intersection nearby the engineering tower at CU.

Of note: the entire implementation of the shim layer to protect these two components included a total of 130 lines of code - a modest amount for the level of control obtained. The total extra memory usage is 1 kilobyte used to store our configuration file, the minimum file size allowed on the system. The MyShield application responsible for managing applications and Security Circles is 719 lines, but only 28 kilobytes of memory and 20 kilobytes of storage.

In order to demonstrate that our system behaves as intended, we placed a copy of Google Maps in each of the three circles and observed its behavior when supplied with location coordinates. In figure 4a, the Maps is in the Trusted circle, and the precise latitude and longitude results in the marker being placed exactly where we expect - as it would if the system did not exist. When Maps is placed into the Trust a Little circle (figure 4b), the marker changes location, despite there being no change in the original input. This is the result of leaving only the first digit after the decimal intact - what we consider to be city level anonymity. Finally, in figure 4c, the location is completely obfuscated and the marker is placed in the Atlantic Ocean off the coast of Africa.

While it is unrealistic to expect that a user would install a mapping application and then deny it access to location data, it serves as an effective demonstration that the system's behavior is as intended, and the Maps application has not ceased to function due to our changes.

We tested the results of altering the device's ID number by installing a game available on the Android Market called Sudoku Puzzles by AndroidDev Team (figure 5). The app requires the user to allow access to device ID in order to differentiate between users who have purchased an ad-free version of the game and those who have not. If the user has not purchased the premium version of the game, then there is no need to give an authentic, or even consistent, serial number to the app. Despite the changing ID produced by placing the app in the Untrusted circle, the app functions as normal. If the user desired to purchase and use the ad-free version of the app, they would first have to place it into either the Trust a Little or Trusted circles so that a consistent device ID could be returned to the application. In our current implementation, if the user were to purchase the ad-free version while the app was in the Trust a Little circle, and then uninstall and reinstall the app, the device ID would change, and the behavior obtained through the purchase would be lost. In future implementations, we may allow for recording a device ID to be used in the event an app is deleted and later reinstalled.

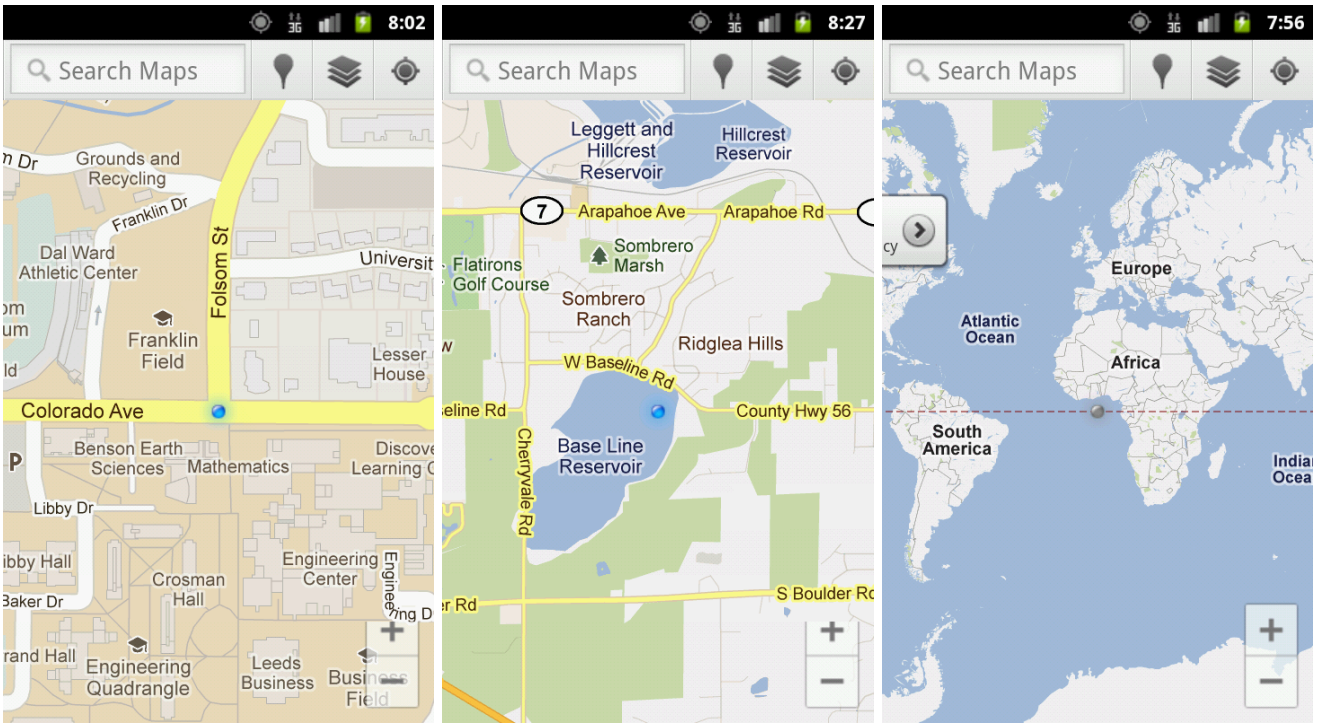
As a more concrete demonstration of the system behavior, we include a screenshot from an application we built specifically for reading the device ID and current location and presenting them in text form. As shown in figure 6, the device ID is generated randomly in both the Trust a Little and Untrusted circles, while the location coordinates become progressively more incorrect moving from the Trusted to Untrusted circles.

We were unable to demonstrate a complete implementation of anonymization of contacts data. The amount of time and code required to mock a database query and response, and to intercept the application requests, was greater than the amount of time available. The Android OS class in question has many methods for providing not only string data, but also binary data for photos. We were able to insert logging statements that do show where the appropriate intercept in the Android OS code is needed. One example of a log entry generated when the default phone dialer system application was used appears as follows: "12-15 19:20:26.695: D/Security Circles - ContactsProvider(199): Checking restricted raw contacts data access for uid 10004".

## 9 Discussion and Future Work

Although we spent most of our time familiarizing ourselves with the Android source and implementing the pieces of our shim layer, we believe that some of the most interesting questions to tackle moving forward will be related to anonymization for various kinds of data and sensors. While our system relies on predefined methods for anonymizing data, it is conceivable that a framework could be created to allow for the insertion of various black box data manipulation subroutines. This would allow other developers to extend our system without having to develop a deep understanding of the operating system, and users could benefit from the effort of several people without having to commit to any particular solution. We envision these alternate anonymization strategies could be deployed via applications available in the Android Market where a system can be taken advantage of for rating applications.

A production version of our system would necessarily incorporate more complete control over data containing and recording pieces of the system. Although we believe our

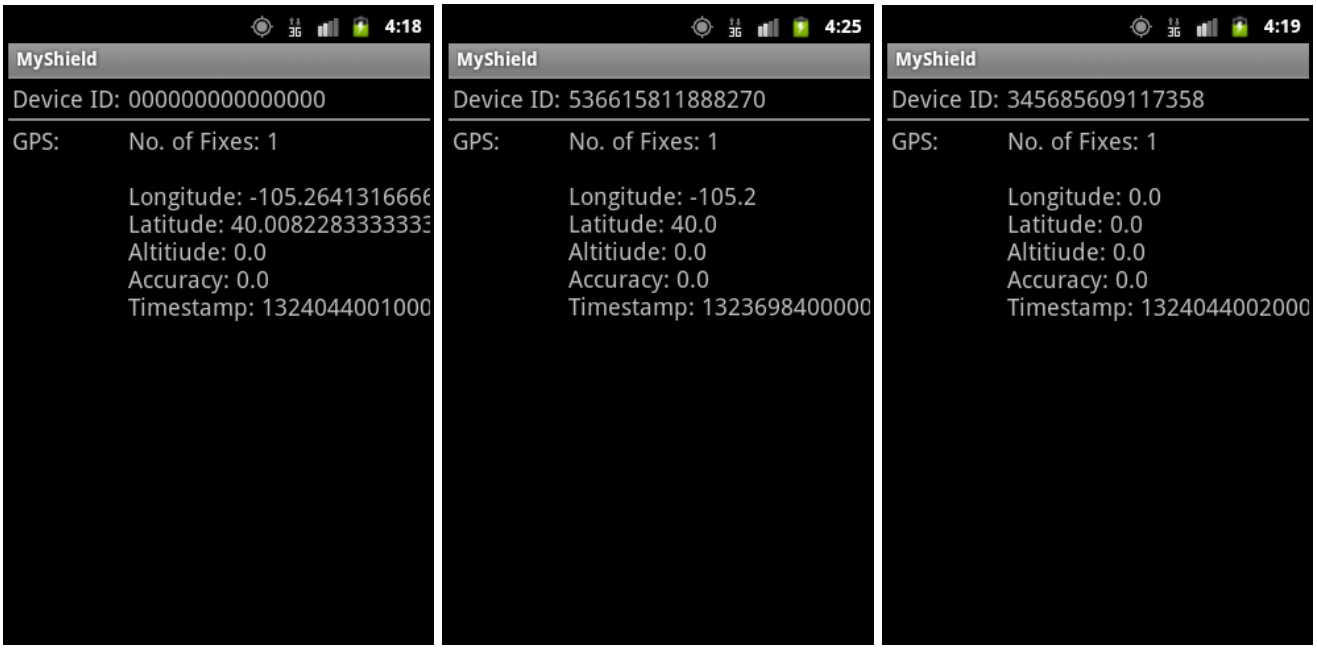



(a) Most Trusted circle

(b) Trust a Little circle (marker still within Boulder)

(c) Do Not Trust circle

Figure 4: Google Maps running in each of the three Security Circles




(a) Trusted circle

(b) Trust a Little circle

(c) Untrusted circle

Figure 6: Demo Application running in each of the three Security Circles

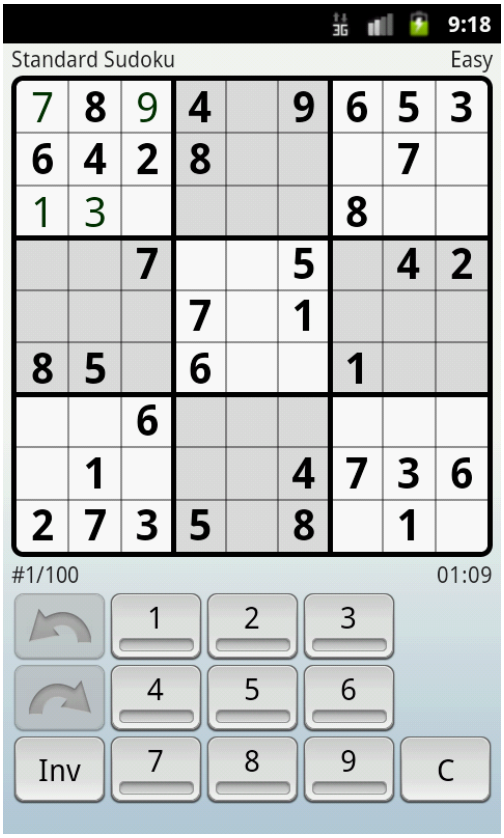



Figure 5: Game in Do Not Trust circle. Even though the ID used is anonymous, the game continues to work

system can be extended to any part of the operating system responsible for disclosing personal information, the task is time consuming and represents diminishing returns from a research perspective. While continuing to develop the shim layer for additional sensor and data interfaces, we believe that we can expedite the process by releasing our changes as part of an open source Android project and encouraging other interested developers to contribute. This also makes more realistic the idea that our system could be kept up to date as new versions of Android are released. In the future we hope to extend the idea of Security Circles in a customizable fashion. Depending on the function of the application, it may be appropriate to restrict access to some data, while allowing unrestrained access to other data. While it may be possible to generate a large number of Security Circles anticipating a variety of situations, we believe that a more intuitive interface would allow the user to create new circles where access can be configured for different data avenues with varying policies.

As part of our continuing evaluation of the system, we plan to implement logic for logging the number of calls to different sensor interfaces. This will allow us to gain a better understanding of what applications are attempting to access what data, and how often those accesses are being performed. We also plan on automating this evaluation so that the results may be presented to the user. In this way the user will be able to have direct insight into the precise behavior of the applications they have installed.

During development of this system, we wondered why it is that, to the best of our knowledge, no large smartphone OS developer has implemented a system similar to the one presented here. Clearly Google has the funding and knowledge required to surpass our effort, but there is no indication that anything like that is forthcoming. One motivation may be that application developers have some basic guarantees about how their application will behave under the current permission structure. Our system eliminates many of these guarantees, and takes a not insignificant amount of control away from developers. Google has several motivations for attracting more developers, so it is unsurprising that they have maintained a simpler system where users are required to conform to accepting a strict set of permissions defined by the developer.

Currently, some sensors available to developers, such as accelerometers, require no permission from the user to be used, despite possibly revealing information about the user's activities [16]. MyShield has the potential to alter this paradigm, although solutions for anonymizing things like accelerometer data are open questions. We believe this may be a fruitful area for future research, made easier by the ability to build on the foundation we have laid with MyShield.

Other sensors of interest for anonymization along with potential strategies:

- Camera - The possibility that an application could misuse the camera to take pictures unknown to the user is one concern. Some applications need this functionality such as social network applications that allow the user to upload pictures to their social network space. One concern with taking pictures on a phone is that many phones will embed GPS location coordinates into the picture file attributes. While this data is useful it may be desirable to remove that information. Another possible aspect of anonymization would be to blur faces in photos. For untrusted applications it may also be appropriate to simply provide a fake image.
- Microphone - For applications that are "trust a little" one could anonymize the voice of the person speaking into the microphone by distorting it. For untrusted applications random tones could be provided instead of actual microphone input so as to fool the application into thinking it is getting data. Another solution may be to provide a more obvious icon or indicator on the screen when the microphone is activated by an application.
- SDCard - Any data stored on the external storage of an Android device does not have the same permissions as applications on internal storage. This is because the common format for external media is FAT which doesn't support Linux permissions. While OS solutions could be provided (virtual file of permissions for example) they are not currently available. For applications that are not trusted one way to anonymize the SDCard usage so an application can't access anything on the card would be to hook in virtual loopback images

instead. This would give an isolated container filesystem to the application that it simply sees as the external storage. The disadvantage is that applications wishing to share data (such as music for example) could not do so. One possible solution is to allow groups of applications to share a loopback file system. Additionally a loop-back file system could be in a format that supports the native OS permission scheme.

- Compass - For "trust a little" anonymization one could restrict the accuracy to simply N, E, S, W. For untrusted one could provide random changing direction information or simply always state N.
- Temperature - One example of a desire to anonymize this sensor would be if a person was being tracked as to being inside or outside. For example, if the outside temperature is known to be below freezing then an application could infer that a user was inside or outside based on the temperature reading of the device. One way to anonymize this data is to simply fix the temperature at some value.
- Light - Similar to temperature the sensor could be use to make inferance about where the user is. A similar method of providing a fixed value or no value would anonymize the user.
- Time/Date - An application could track a user as they move from one time zone to another while traveling. In addition the time of day (day/night) could be used to infer if the user were asleep or not (especially with combination of the accelermator). Anonymizing this data could be accomplished by returning a time of day that stays within a fixed range of hours (say daylight hours only). If the application relied on comparing timestamps it could incur eratic behavior. Another option would be to simply slow time down so that each request for the current time returns a timestamp incremented by 1 second,minute, etc. instead of the real time.
- Contacts - As described in previous sections providing an anonymized list of contacts will be very useful. Future work would involve the full implementation of such.

MyShield does not address applications developed using native code and the Android Native Development Kit. In the interest of developing a working system, we chose to implement our shim layer code at a level that native code may be able to bypass.

## 10 Conclusions

This paper has demonstrated the feasibility and implementation of managing application permissions through an intuitive Security Circles interface coupled with a low level operating system shim layer. This solution improves upon previous methods by eschewing the requirement for an application operating with elevated privileges, and allowing applications to continue operating through our transparent shim layer. This paper also discusses methods for anonymization of personal data in a way that allows applications to continue to be useful to the user while avoiding disclosure of sensitive data.

## 11 References


[1] 3GPP2. 3G Mobile Equipment Identifier. http://www.3gpp2.org/Public_html/specs/S.R0048-A_v4.0_050630.pdf, 2005.

[2] Apple. iOS Operating System. http://www.apple.com/ios, 2007.

[3] GSM Association. IMEI Allocation and Approval Guidelines. http://www.gsmworld.com/documents/DG06_v5.pdf, 2010.

[4] Tim Bray. Be Careful With Content Providers. http://android-developers.blogspot.com/2010/05/be-careful-with-content-providers.html.

[5] Canalys. Google's Android becomes the world's leading smart phone platform. http://www.canalys.com/newsroom/google's-android-becomes-world's-leading-smart-phone-platform, 2011.

[6] Cory Cornelius, Apu Kapadia, David Kotz, Dan Peebles, Minho Shin, and Nikos Triandopoulos. Anonysense: Privacyaware people-centric sensing. In *In Proc. ACM 6th Intl Conf. on Mobile Systems, Applications and Services (MOBISYS 08*, 2008.

[7] Elinor Mills. Malicious Android apps double in six months. http://news.cnet.com/8301-27080_3-57342661-245/malicious-android-apps-double-in-six-months, December 2011.

[8] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. OSDI.

[9] Google. Android Open Source Project. http://source.android.com.

[10] Google. Android Operating System. http://www.android.com, 2007.

[11] Google. Google+ Social Network. http://plus.google.com, 2011.

[12] Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 31–42, New York, NY, USA, 2003. ACM.

[13] Steve Kondik. CyanogenMod. http://www.cyanogenmod.com/.

[14] Lamian. LBE Privacy Guard. http://market.android.com/details?id=com.lbe.security.lite, November 2011.

[15] Ling Liu. Privacy and location anonymization in location-based services. *SIGSPATIAL Special*, 1:15–22, July 2009.

[16] Emiliano Miluzzo, Nicholas D. Lane, Kristof Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application. ACM, 2008.

[17] Heechang Shin, Vijayalakshmi Atluri, and Jaideep Vaidya. A profile anonymization model for privacy in a personalized location based service environment. In *Proceedings of the The Ninth International Conference on Mobile Data Management*, pages 73–80, Washington, DC, USA, 2008. IEEE Computer Society.

[18] Akiyoshi Suzuki, Mayu Iwata, Yuki Arase, Takahiro Hara, Xing Xie, and Shojiro Nishio. A user location anonymization method for location based services in a real environment. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 398–401, New York, NY, USA, 2010. ACM.