BlurSense: Dynamic Fine-Grained Access Control for Smartphone Privacy

Justin Cappos[†], Lai Wang[†], Richard Weiss[*]
Yi Yang[◁], Yanyan Zhuang[†‡]
[†]NYU Poly, [‡]University of British Columbia
[*]Evergreen State College, [◁]Fontbonne University

***Abstract*—For many people, smartphones serve as a technical interface to the modern world. These smart devices have embedded on-board sensors, such as accelerometers, gyroscopes, GPS sensors, and cameras, which can be used to develop new mobile applications. However, the sensors also pose privacy risks to users. This work describes BlurSense, a tool that provides secure and customizable access to all of the sensors on smartphones, tablets, and similar end user devices. The current access control to the smartphone resources, such as sensor data, is static and coarse-grained. BlurSense is a dynamic, fine-grained, flexible access control mechanism, acting as a line of defense that allows users to define and add privacy filters. As a result, the user can expose filtered sensor data to untrusted apps, and researchers can collect data in a way that safeguards users' privacy.**

Keywords*—*Sensor access control, privacy, data filtering

I. Introduction

Smartphones and tablets are becoming the dominant way that people interact with the physical world. This trend is showing no sign of stopping. Smartphones overtook PC sales in 2011, and even tablets will outsell desktop PCs by 2013 [1]. As of October 2012, there were more than one billion smartphones in use [2]. By the end of 2013, the number of mobile devices (like smartphones and tablets) is projected to surpass the number of people on the planet [3]. Given such ubiquity across demographic and geographic spectrum, smartphones can become important tools for scientific research in addition to their role in personal communication.

Smartphones and tablets in the modern age have powerful capabilities in computing, communication, and sensing [4]. Like desktop and laptop machines, they can perform complex computing tasks according to user commands. These smart devices can also communicate with each other through wireless communications. Unlike regular computers, they have a rich set of onboard sensors, such as accelerometers, gyroscopes, GPS, and cameras, which enable the development of innovative mobile applications [5]. Although the sensing capabilities enhance the convenience of user interfaces and application usefulness, they also raise serious privacy concerns [6]. For instance, through accessing sensor data, malicious applications could retrieve sensitive information about the mobile phone users, such as location, passwords, and credit card numbers [7], [8], [9], [10]. They even might be able to send these sensitive information to remote attackers [11], [12]. There has been alarming news about privacy breaches of personal data on smart devices: 26% of Android apps in Google Play can access user's personal data [13]; an iOS app auto-posts false piracy accusations on users' Twitter accounts [14]; apps can steal sensitive information like passwords using the smartphone's motion sensors to determine tapped keys [7]; and a huge botnet that is collecting sensor data was discovered on more than a million end user smartphones [15]. The Federal Trade Commission (FTC) recently recommended that mobile platforms should provide in-time disclosures to users of accessing sensitive content on smart devices [16].

The current access control to the smartphone resources, such as sensor data, is static and coarse-grained. However, such defense is pre-determined by the manufacturer. Take the Android platform as an example, the access permissions are either granted or denied completely during the installation of applications based on a request XML manifest file. As a result, applications may ask for more permissions than are actually required for operation. Having been granted the requested permissions, applications have access to those resources permanently. Some systems have been proposed to address this issue; however, they require modifications to the Android platform [17], [18]. This increases the cost of maintenance, is less flexible and cannot be used in legacy systems. In addition, the user would need to trust that the new operating system is not malicious and is not more vulnerable than the standard one. With BlurSense there is less risk; the worst that can happen is that the app will have the same access to the sensor data that is permitted by the manifest file. To protect smartphone users' privacy, we propose a *second line of defense*: a dynamic, fine-grained, flexible access control mechanism, which incorporates privacy filters in user space [19], [20]. We want to let users choose defenses freely in a marketplace so that different vendors can build them.

Specifically, we will implement a framework of reference monitors, to enforce mandatory access control to sensor data in real time. A reference monitor is a method or function that implements an access control policy for a set of resources and is usually specified in terms of what capabilities are allowed. The access control that we are concerned with in this paper is for sensor data with respect to applications in user space. If such an application needs to access any of the sensor data, our second line of defensive (reference monitors) will come into play, mediating every access to sensor data. As a result, a user should be able to combine solutions from different vendors. If a vendor's product is ineffective or the vendor is malicious, the user can still be protected.

According to the semantics of the access requests and the current context, when a remote procedure call is made to the Android OS to request sensor data, it will be handled by a reference monitor. Based on a sensitivity of the application, the data returned may be filtered, dropped or passed through. Note that the reference monitor cannot pass on data that is blocked by the manifest file because it is layered on top of whatever privacy and security mechanisms that are in place at the OS level. If the request is for highly sensitive sensor data, then the request might be simply rejected. Otherwise, if the request is for medium-level sensitive sensor data, the request might get through, but the returned data has reduced resolution. If the request is for low-sensitive sensor data, then the return results need to be processed, e.g., by filtering or obfuscation. BlurSense can be used with Sensorium [21], which provides a rich and extensible collection of sensor data from Android devices.

II. System Architecture and Threat Model

In this section, we first briefly describe Sensorium, the basis of BlurSense. Then we discuss the new extensions to Sensorium for enabling BlurSense through defining a threat model.

A. Building Sensorium

For BlurSense to work, it must have access to the sensor data. However, different devices and platforms, such as Android and iOS, use very different interfaces for their sensors. One of the major goals of our testbed is to create a uniform interface to support a wide range of sensor categories, broader device and network diversity, while the client software still behave in a portable manner.

1) Seattle Porting onto Mobile Devices: Seattle is the testbed platform we have developed over the past four years [22]. It supports a wide range of devices including desktop, laptop, servers, etc. We recently ported Seattle onto mobile platforms.

Compared to desktop and laptop environments, development on mobile platforms has more resource limitations, such as limited computational power and battery levels. However, researchers were able to do early stage ports of Seattle to Android [23] (and jailbroken iPad / iPhone / iPod) with a few weeks of developer effort. Users can now download a native Android installer (APK) from the Google Play store [24]. Our Seattle testbed on Android supports Android versions from 2.1 to 4.0.4 (API levels 7 to 15), covering device versions with the highest market distribution [25]. Despite never being advertised or mentioned publicly, our Seattle app in Google Play has more than 50 installs.

2) Supporting Sensors on Mobile Devices: Due to its isolation of the VM, Seattle apps cannot normally access sensor and other data on a user's mobile device, such as GPS, WiFi SSID and signal strength, motion sensors (e.g., accelerometer, compass, gyroscope, barometer), etc. However, this data if anonymized would be very useful to researchers. Sensorium provides an API for reading sensor data from a Seattle VM and providing it to apps for research. Sensorium is a generic sensor reading framework built on top of Seattle on Android. It funnels data from actual sensor drivers, implements fine-grained privacy control for the user, and provides generic outbound interfaces such as XML-RPC. However, the APIs provided by native smartphone sensors vary significantly across platforms. Our philosophy is to provide a simple API that would allow a variety of sensor applications to operate in a unified manner.

First, we implemented system hooks called *sensor modules* to interact with a variety of sensors through system programming interfaces. Currently, implemented sensor modules and the available contextual information are classified into three categories: device specific (percentage of battery power level, CPU and memory utility), location related (latitude, longitude, altitude, accuracy, and speed if available), and network related (mobile network type and operator, nearby WiFi access point and Bluetooth devices). While sensor modules are the system hooks with read access to valuable sensor resources, they cannot manipulate sensor data. Additionally, the sensor API also provides a base *registry service* with a common interface for use by a sensor implementation. For both local and remote processes to access sensor data, an XML-RPC library [26] is incorporated to provide data in an unified format. In case newer sensors appear on future mobile devices, developers can add newly implemented sensors into this framework rather easily. The registry service listens for connection on a set of predefined ports via XML-RPC. Thus, both local and remote process can connect to these ports and register for sensor updates.

Our preliminary work in this area has resulted in working code [21], tutorials [27], and a blog for problem discussion [5]. Several different groups have already used our early-stage proof-of-concept to solve problems across a variety of domains, demonstrating the potential of sharing sensor data.

B. BlurSense Threat Model

While sensors on smartphones have many useful applications, they also pose a risk to users. A study showed that unscrupulous hackers typically find personal information stored on devices inviting [28]. To further motivate our work, we consider three cases in our threat model. The first category under consideration is called **greedy legitimate applications**. In this category, the developers of the applications ask for more permissions in the manifest file than are needed, e.g. to increase revenue. For example, some applications run hidden code, connecting to remote advertisement servers, which analyze smartphone users' behavior.

The second category is called **compromised legitimate applications**. In this category, the legitimate applications have certain vulnerabilities, such as buffer overflow, so these applications are under the full control of remote attackers. The attackers might take the advantage of liberal access policies for the application to collect sensitive information about the users, such as contacts, credit card numbers, and passwords, to jeopardize users privacy.

The third category is called **malicious applications**. These applications are designed by attackers with malicious purposes. They behave like benign applications. However, they covertly collect users private information, like trojans, and send these information to the remote attackers.

In the following section, we introduce our proposed scheme that is able to detect and defend against all these three types of threats above.

III. BLURSENSE DESIGN

A. BlurSense Overview

1) Overview of the Solution: Today's smartphone OS typically exposes resources based on a static policy. Such permissions are often much more than necessary. Several related work has been proposed to refine or reduce permissions on mobile platforms [18], [29], [17], [30] via modifying the device platform. BlurSense allows untrusted parties to add privacy filters from user space. Multiple security vendors can efficiently and effectively collaborate to strengthen user privacy.

Different from the existing privacy controls, BlurSense provides a programmable privacy protection framework. Users not only gain full transparency of what information is captured on their devices, but also have full control over how much information they would share with the rest of the world — a secure personal data ecosystem. After installing BlurSense, a user can install software from a third party (like a security vendor) that performs custom sensor filtering actions in response to application requests. For example, an application could be prevented from using motion sensors when running in the background, or precise GPS data could be abstracted to a neighborhood or zip code. BlurSense provides effective controls for smartphones, much like Flash and JavaScript filtering tools protect laptops and desktops (e.g., NoScript [31], AdBlock [32], FlashBlock [33]).

2) What BlurSense Provides: Seattle provides a sensor interposition mechanism and a sandboxing mechanism that make it easy to implement privacy filters. A user can let a third party have access to their sensor data from within a security and performance isolated container [20]. By leveraging this security, the user provides minimal trust in the third party, but allows them an easy way to code their filters. This functionality also automatically handles multiple privacy filters from different parties through the sandbox policy composition functionality [20].

Researchers who use BlurSense will build a mechanism to trap the requests from generic applications and pass them into BlurSense (a proof-of-concept for Android has already been built). They will build and manage an "App Store" for BlurSense to allow users to locate privacy filters they wish to apply. These may range from sharing sensor data with researchers by reducing the precision of sensor values, salting and hashing sensor values for anonymizing collected data, or completely denying access to individual (or all) sensors. For a particular sensor, a filter might perform an action such as blurring the resolution of photos and video taken by the camera, removing access point information from WiFi scans, or omitting the motion sensor data completely. Security and privacy groups can easily build and disseminate their own privacy filters they recommend to users by adding them to BlurSense app store. Therefore, BlurSense is able to handle the three categories of threats in Section II-B.

B. Design Framework

As shown in Figure 1, we design a user GUI interface, so that users can configure the policies used by our framework of reference monitors. For example, the users can define different blur thresholds. The users can also assign some initial sensitivity scores for different sensors and operations. Under the control of policy management, our framework takes those permissions which are allowed by the Android reference monitor [34], [35]. If these permissions access to sensor data, then the framework will accept them as inputs. Based on the defined policies from the user interface application, the framework takes different actions, blocking the requests or returning original/blurred data.

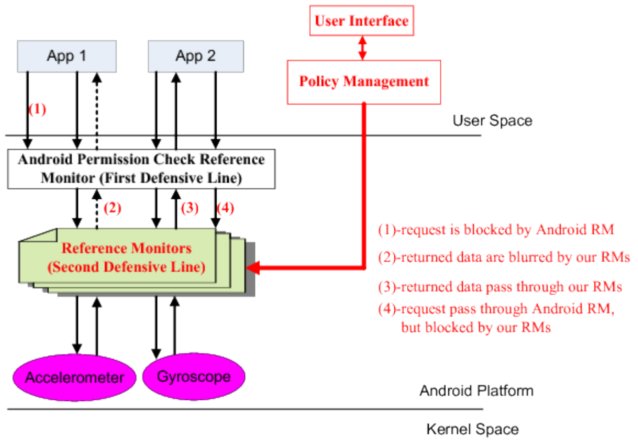
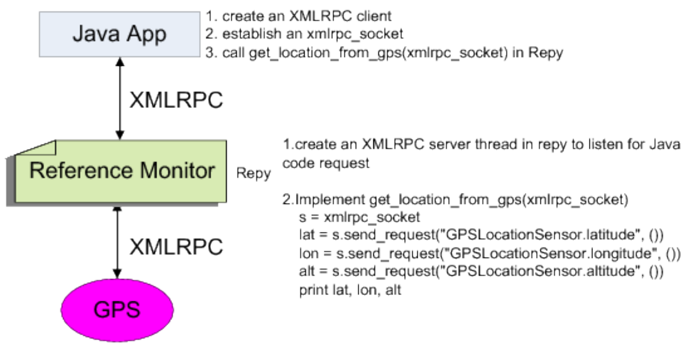

User Interface
App 1
App 2
Policy Management
User Space
(1)
Android Permission Check Reference Monitor (First Defensive Line)
(2)
(3)
(4)
Reference Monitors (Second Defensive Line)
(1)-request is blocked by Android RM
(2)-returned data are blurred by our RMs
(3)-returned data pass through our RMs
(4)-request pass through Android RM, but blocked by our RMs
Accelerometer
Gyroscope
Android Platform
Kernel Space

Fig. 1: System architecture.

C. Implementation of Framework



```
1. create an XMLRPC client
2. establish an xmlrpc_socket
3. call get_location_from_gps(xmlrpc_socket) in Repy
```

```
1.create an XMLRPC server thread in repy to listen for Java
code request

2.Implement get_location_from_gps(xmlrpc_socket)
   s = xmlrpc_socket
   lat = s.send_request("GPSLocationSensor.latitude", ())
   lon = s.send_request("GPSLocationSensor.longitude", ())
   alt = s.send_request("GPSLocationSensor.altitude", ())
   print lat, lon, alt
```

Fig. 2: A preliminary implementation.

Figure 2 shows an example of implementing a GPS sensor. There are three different components: a Java native app on Android, the reference monitor and the sensor. In this case, the sensor provides GPS coordinates of a device. The Java native app requests sensor data as an XML-RPC client. The request is then interposed by the reference monitor, which is both an XML-RPC server and client. As a server, it listens for incoming XML-RPC requests from the native app, while it also requests sensor data from Sensorium. As raw sensor data is returned by Sensorium, the reference monitor blurs the data according to its different blurring policy defined for each threat model, as in Section II-B. As a result, BlurSense is able

TABLE I: Performance overhead of BlurSense

	Latitude	Longitude	Altitude
Median (second)	0.2704	0.2575	0.5017
Std (second)	0.1259	0.0718	0.0

to handle the three categories of threats by applying different blurring policies.

D. Performance Overhead

Table I shows the performance overhead of BlurSense when a native app requests GPS sensor data, as in Figure 2. Because the altitude of the user was not changed, the value for altitude is the result from one measurement with 0 standard deviation. The values for latitude and longitude are collected from 22 measurements. As seen from the table, the overhead of BlurSense is lower than 0.3 second for latitude and longitude, and their variation is quite low. The value for altitude here is not statistically meaningful. Overall, the overhead is not perceptible.

IV. Conclusion and Future Work

Smartphones have become more and more popular and indispensable for users nowadays. Sensors in smartphones facilitate the development of innovative mobile applications, however, they also raise people's concerns in exposing users' privacy. The goal of BlurSense project is to provide interfaces in user space so that privacy filters could be customized and developed to protect accesses to sensor resources. In this way, smartphone users' privacy could be protected in a dynamic, fine-grained, and flexible way. This paper describes the detailed design of the framework and a preliminary implementation. In the future, we will work on completing the implementation and obtaining data of performance evaluation on side effects and performance penalties such as latency and power consumptions.

References

[1] "Smartphone sales overtook PCs in 2011," http://www.neowin.net/news/smartphone-sales-overtook-pcs-in-2011.

[2] "Number of smartphones tops one billion," http://www.telegraph.co.uk/finance/9616011/Number-of-smartphones-tops-one-billion.html.

[3] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20122017," http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

[4] L. Cai, S. Machiraju, and H. Chen, "Defending against sensor-sniffing attacks on mobile phones," in *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*. ACM, 2009, pp. 31–36.

[5] "Sensibility Testbed," http://seattlesensor.wordpress.com/.

[6] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *Security & Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010.

[7] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012, pp. 113–124.

[8] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 323–336.

[9] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: a new video-based spyware in 3g smartphones," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 69–78.

[10] L. Cai and H. Chen, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX conference on Hot topics in security*. USENIX Association, 2011, pp. 9–9.

[11] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones." in *NDSS*, vol. 11, 2011, pp. 17–33.

[12] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 551–562.

[13] "More Than 25% of Android Apps Know Too Much About You," http://yro.slashdot.org/story/12/11/02/1316238/more-than-25-of-android-apps-know-too-much-about-you.

[14] "App Auto-Tweets False Piracy Accusations," http://yro.slashdot.org/story/12/11/13/2249203/app-auto-tweets-false-piracy-accusations.

[15] "China mobile users warned about large botnet threat," http://www.bbc.co.uk/news/technology-21026667.

[16] "US Wants Apple, Google, and Microsoft To Get a Grip On Mobile Privacy," http://yro.slashdot.org/story/13/02/02/2124204/us-wants-apple-google-and-microsoft-to-get-a-grip-on-mobile-privacy.

[17] M. Conti, V. T. N. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security*. Springer, 2011, pp. 331–345.

[18] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 639–652.

[19] "BlurSense," https://blursense.poly.edu.

[20] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson, "Retaining sandbox containment despite bugs in privileged memory-safe code," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 212–223. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866332

[21] "Seattle Sensors, GitHub," https://github.com/fmetzger/android-sensorium.

[22] "Seattle," accessed October 4, 2013, https://seattle.poly.edu/.

[23] "Seattle on Android – Seattle – Trac," https://seattle.poly.edu/wiki/SeattleOnAndroid.

[24] "Seattle Testbed on Android," https://play.google.com/store/apps/details?id=com.seattleonandroid.

[25] "Platform Versions," http://developer.android.com/about/dashboards/index.html.

[26] "android-xmlrpc: Very thin xmlrpc client library for Android platform," https://code.google.com/p/android-xmlrpc/.

[27] "Seattle Sensors Project," http://seattlesensors.poly.edu/.

[28] D. Mulligan and A. Schwartz, "Your place or mine?: privacy concerns and solutions for server and client-side storage of personal information," in *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*. ACM, 2000, pp. 81–84.

[29] Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming information-stealing smartphone applications (on android)," *Trust and Trustworthy Computing*, pp. 93–107, 2011.

[30] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of AsiaCCS*, 2012.

[31] "NoScript," http://noscript.net/.

[32] "AdBlock," http://safariadblock.com/.

[33] "Flashblock," http://en.wikipedia.org/wiki/Flashblock.

[34] W. Enck, M. Ongtang, and P. McDaniel, "Understanding android security," *IEEE Security and Privacy*, 2009.

[35] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, "Semantically rich application-centric security in android," in *ACSAC*, 2009.