ipShield: A Framework For Enforcing Context-Aware Privacy

Supriyo Chakraborty, Chenguang Shen, Kasturi Rangan Raghavan, Yasser Shoukry, Matt Millar, and Mani Srivastava, ***University of California, Los Angeles***

https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/chakraborty

This paper is included in the Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14).

April 2–4, 2014 • Seattle, WA, USA

ISBN 978-1-931971-09-6

Open access to the Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14) is sponsored by USENIX

ipShield: A Framework For Enforcing Context-Aware Privacy

Supriyo Chakraborty, Chenguang Shen, Kasturi Rangan Raghavan,
Yasser Shoukry, Matt Millar, Mani Srivastava
University of California, Los Angeles

Abstract

Smart phones are used to collect and share personal data with untrustworthy third-party apps, often leading to data misuse and privacy violations. Unfortunately, state-of-the-art privacy mechanisms on Android provide inadequate access control and do not address the vulnerabilities that arise due to unmediated access to so-called *innocuous* sensors on these phones. We present ipShield, a framework that provides users with greater control over their resources at runtime. ipShield performs *monitoring* of every sensor accessed by an app and uses this information to perform privacy *risk assessment*. The risks are conveyed to the user as a list of possible inferences that can be drawn using the shared sensor data. Based on user-configured lists of allowed and private inferences, a *recommendation* consisting of binary privacy actions on individual sensors is generated. Finally, users are provided with options to override the recommended actions and manually configure context-aware *fine-grained* privacy rules. We implemented ipShield by modifying the AOSP on a Nexus 4 phone. Our evaluation indicates that running ipShield incurs negligible CPU and memory overhead and only a small reduction in battery life.

1 Introduction

Smartphones have evolved from mere communication devices into sensing platforms supporting a sprawling ecosystem of apps which thrive on the continuous and unobtrusive collection of personal sensory data. This data is often used by the apps to draw inferences about our personal, social, work and even physiological spaces [10, 41, 16, 53, 9, 58, 49, 51] often under the pretext of providing personalized experiences and customized recommendations. However, not all app developers are equally trustworthy, and this coupled with user naïveté leads to data misuse and privacy concerns.

To safeguard user privacy, Android requires developers to specify the permissions needed by their apps. At install time, a user can either grant access to all the requested resources or opt to not use the app at all. But despite these provisions, cases of privacy violations by third-party apps are rampant [33, 6, 55]. We observe multiple problems with the current privacy mechanisms in Android. First, only a select set of sensors such as GPS, camera, bluetooth are considered to be privacy-prone and have their access mediated through protected APIs [3]. Other onboard sensors such as accelerometer, gyroscope, light, etc. are considered to be innocuous, requiring no user permission. This specific vulnerability of unrestricted access to accelerometer and gyroscope data has been exploited to mount keylogging attacks [43], and for reconstruction of travel trajectories [31]. Second, various studies [52, 28], to understand users' perception of privacy in general and their understanding of Android permissions in particular, reveal that users are often oblivious to the implications of granting access to a particular type of sensor or resource on their phone at install time. However, the perception quickly changes to one of concern when apprised of the various sensitive inferences that could be drawn using the shared data. Finally, users only have a binary choice of either accepting all the requested permissions or not installing the app at all. Once installed, users do not have any provision to revoke or modify the access restrictions during runtime.

Prior research have tried to address some of the above problems. TaintDroid [24] extends the Android OS by adding taint bits to sensitive information and then tracking the flow of those bits through third-party apps to detect malicious behavior. However, tainting sensor data continuously for all apps has high runtime overhead, and is often conservative as data sensitivity typically depends on user context. Moreover, TaintDroid stops at detection and does not provide any recommendation on countering the privacy threat. MockDroid [18] is a modified Android OS designed to allow users the ability to mock resources requested by the app at runtime. Mocking is used to simulate the absence of resources (e.g., lack of GPS

fix, or Internet connectivity), or provide fixed data. However, MockDroid only works for resources explicitly requested by an app (innocuous sensors are not handled), is binary because a user can either mock a resource or provide full access to it and finally MockDroid falls short on providing any guidance to the user regarding which sensors to mock. PMP [12] is a system that runs on iOS and allows users to control access to resources at runtime. It uses a crowdsourced recommendation engine to guide users towards effective privacy policies. However, PMP does not handle sensor data.

In this paper, we present ipShield [5], a privacy-enforcing framework on the Android OS. ipShield allows users to specify their privacy preferences in terms of semantically-meaningful inferences that can be drawn from the shared data, auto generates privacy actions on sensors, and supports an advanced mode for manual configuration of fine-grained privacy rules for accessed sensors on a per app basis at runtime. We build on prior work in [7, 18] and make the following contributions.

- We modified the Android OS to monitor all the sensors accessed by an app regardless of whether they are specified explicitly (e.g., in the manifest file for Android apps) at install time. As per our knowledge, ours is the first system that tracks innocuous sensors.
- We took an important step towards presenting the privacy risks in a more user-understandable format. Instead of listing sensors, we list the inferences that could be made using the accessed sensors. Users can specify their privacy preferences in the form of a prioritized blacklist of private inferences and a prioritized whitelist of allowed inferences.
- We implemented a recommendation engine to translate the blacklist and the whitelist of inferences into lower-level binary privacy actions (suppression, allow) on individual sensors.
- Finally, we provided the user with options to configure context-aware fine-grained privacy actions on different sensors on a per app basis at runtime. These actions range in complexity from simple suppression to setting constant values, adding noise of varying magnitude, and even play-back of synthetic sensor data.

ipShield is open source and implemented by modifying Android Open Source (AOSP) [2] version 4.2.2_r1. We evaluated it using computation intensive apps requiring continuous sensor data. Our results indicate that ipShield has negligible CPU and memory overhead and the reduction in battery life is $\sim 8\%$ in the worst case.

2 Case Studies

Using two typical scenarios we illustrate below how ipShield can help app users protect their privacy.

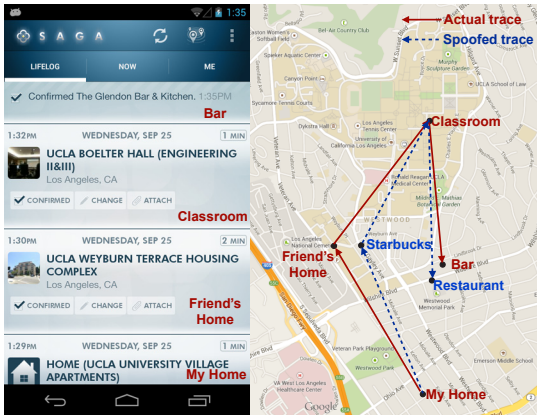

Figure 1: Left: Saga app showing actual trace of the user. Right: Both actual trace and spoofed trace on the map.

2.1 Transportation Mode and KeyLogging: Accelerometer/Gyroscope

Activity recognition algorithms [16, 53] are used by various fitness apps to infer the users' Transportation Mode (e.g., to predict one of three labels: *walking*, *motorized* or *still*). For example, the Ambulation app in [53] combines accelerometer and GPS data to infer the labels with over 90% accuracy. However, the same data can also be used to infer other labels sensitive to the user. For example, accelerometer together with gyroscope data can be used to perform keylogging and to infer keystrokes (Onscreen Taps) on the softkeyboard [43] (and separately to also infer Location [31]) with over 80% accuracy. This leads to the leakage of sensitive information like password and PIN entered on the phone.

Using ipShield, a user would add the Transportation Mode and the Onscreen Taps to the whitelist and the blacklist, respectively. This will block the accelerometer and gyroscope data from reaching the Ambulation app preventing keylogging. However, this will also cause the app to stop performing activity recognition. In Section 8 we will show how ipShield can be used to configure fine-grained rules and maximize the utility of the app.

2.2 Saga: Location

Saga [10] is a life logging app which runs in the background and keeps track of the places visited by a user. By analyzing the location trace of a user, Saga can infer useful information such as average daily commute time, time spent at work, etc. However, it can also derive sensitive inferences about locations such as *home, office, hospital* private to the user. Fig. 1(left) shows a mobility trace recorded using Saga. The user starts from home, picks up her friend and drives to school for class; later she also visits a nearby bar and wants to keep the visit private. In addition to this direct privacy requirement, there is also an indirect privacy concern. Saga

reveals the home location of the user's friend. The location information can be coupled with other online resources to identify the home owner, and infer that the friend had gone to the bar too. Thus, privacy of both the user and her friend is compromised, even though the friend is not using Saga. We therefore want ipShield to allow spoofing of location traces to protect visits to sensitive places. A plausible spoofed trace is shown on the map in Fig. 1(right). We illustrate how ipShield achieves this in Section 8.

3 Inference Privacy Problem

Inferences are labels (of activity, behaviour, places etc.) associated with data. We group labels of a similar (semantic) type into an *inference category*. The category names and the grouping are based on prior work (see Table 3). For example, *hospital, home, office* are grouped under Location category. An adversary tries to infer/predict these labels from the shared data. The prediction accuracy of an inference category corresponds to correctly predicting a label in that category. We now define the inference privacy problem.

Problem statement: Data is typically shared with an app for a specific set of inference categories. For example, in Section 2.1, data is shared for inferring the Transportation Mode, and in Section 2.2 it is for inferring travel statistics. These categories and their labels form a whitelist which the user wants to allow and obtain utility. However, the same data can be used to infer keystrokes and sensitive locations – inferences sensitive to the user. The sensitive categories and their labels form the blacklist which the user wants to keep private. Each inference category can also be associated with a user specified priority level (Section 6.2). The privacy problem is to design a system which will take as input a whitelist and a blacklist of prioritized inference categories and translate them into privacy actions on sensors such that the two lists are balanced as per user specified priorities.

Side-Channel Attacks: Traditionally, side channel attacks are designed to exploit the information revealed by execution of cryptographic algorithms to recover their secret key. These attacks typically use information channels which include but are not limited to running time, cache behavior, power consumption pattern, acoustic emanations and timing information [36, 54, 29] during execution. Sometimes, even with no program execution, information side channels can exist due to physical signals emanating from a hardware while it is being used by a user. For example, acoustic [13, 42] and electromagnetic [57] emanations from a keyboard has been used to infer keystrokes and recover sensitive passwords and PINs. The feasibility of such attacks on the smart phone using sensor data has been demonstrated in [14, 43].

Privacy Analysis	Kirin [25], SOM [17], Stowaway [27]
Privacy Detection	**Static**: BlueSeal [32]
	Dynamic: TaintDroid [24]
Privacy Mitigation	**Mobile Based**: Dr. Android Mr. Hide [34], PMP [12], Apex [45], MockDroid [18], AppFence [33], pDroid [7], πBox [38]
	Cloud Based: Lockr [56], PDV [44], Persona [15]

Table 1: Categorization of prior work.

Our inference privacy problem differs from traditional side-channel attacks in several ways. First, the shared data used for the attack are not unintended physical signals emanated from the hardware, or covert timing information but sensor data intended for the recipient. Second, in our setting the recipient is also the adversary whereas in case of side-channel attacks the adversary is typically different from the intended recipient. Finally, at least in principle, the side-channel attacks can be prevented by placing the computational hardware in physically isolated and secure chamber whose boundaries the electromagnetic, acoustic and such emanations cannot cross which is not the case in our scenario. In [14], inferring the keystrokes is referred as a side-channel attack. However, we call it a blacklist inference as the sensor data are intended to be shared with the app for the whitelisted inferences and are not a side-channel.

4 Related Work

We group prior work on systems for protecting privacy under three broad categories as shown in Table 1. The Privacy Analysis category summarizes contributions towards analysis of the Android permission model, conformance of the various apps to this model, the usage pattern of permissions across apps, and finally the expressibility of the permission model [25, 17, 27]. Under Privacy Detection we have tools such as BlueSeal [32] which use static analysis of the app bytecode to detect if sensitive information is being leaked over the network interface and inform it to the user at install time. Other systems like TaintDroid [24] use dynamic flow tracking to detect malicious app behavior. However, both techniques can only alert the users of malicious behavior (BlueSeal at install time, and TaintDroid at runtime), and do not provide suitable mechanisms to prevent information leakage.

Under the Privacy Mitigation category we group privacy systems implemented on mobile platforms. Dr. Android and Mr. Hide [34] instrument and modify the app's dex bytecode to ensure that access to all private resources is made available only through their trusted interface. AppFence [33] builds on TaintDroid and provides *shadow* or synthetic data to untrusted apps and measures the effect of such data on app utility. Other systems in [18, 7, 45, 12, 38] provide users with the

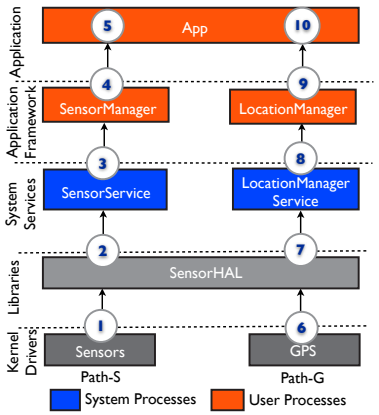

Figure 2: The data flow path from sensors to apps. Same colored blocks represent components within the same process.

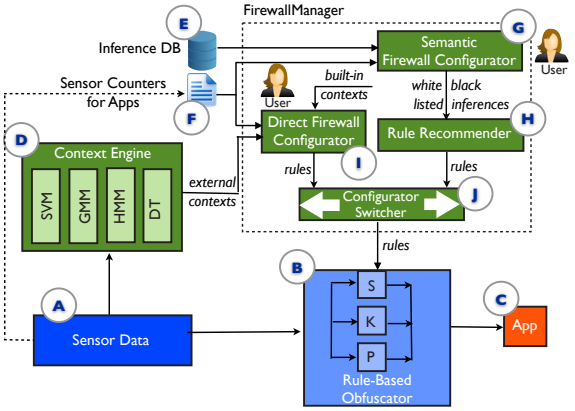

Figure 3: ipShield data flow.

ability to control access to their resources at runtime - a feature that is currently being integrated into the latest Android release [1]. However, the above systems provide binary access control to resources, do not monitor access to innocuous sensors, and lack high-level user-understandable privacy abstractions (inferences). Cloud-based solutions in [56, 44, 15] are for protecting privacy of data streams but require additional infrastructure. A detailed exposition of other various privacy preserving techniques, and initial ideas on ipShield can also be found in [21].

5 Background: Android

Below we describe data paths, from sensors to apps and also highlight Android's security model [3] to understand the process level isolation of the components.

5.1 Android Sensor Data Flow Path

We consider two data paths as shown in Fig. 2. Path-S is used by sensors such as accelerometer, gyroscope, light and so on. Path-G is from the GPS to apps. Note that the paths are simplified representation showing only the components of the Android OS that are relevant to ipShield. `SensorService` and `LocationManagerService` are system services (running continuously in the background) which are started by the Android OS at boot time. These services run as separate threads within the `system_server` process, poll the `SensorHAL` layer for sensor data and are responsible for pushing the data to the apps. The apps typically do not communicate directly with the services. Each system service has a corresponding `Manager` which acts as its proxy. Thus, to access sensor data an app instantiates either a `SensorManager` or a `LocationManager` object and uses the object's public methods to register an event listener for the desired sensor. As shown in Fig. 2, both the app and the manager objects are part of the same process (which also runs the Dalvik Virtual Machine).

5.2 Android Security Model

Application Sandboxing: The core of the Android OS is built on top of the Linux kernel, and this allows Android to re-purpose the traditional security controls built into Linux to protect user data, system resources, and to provide app isolation. Android enforces kernel level *Application Sandboxing* for every software that runs above the kernel which includes all apps, OS libraries, OS-provided app framework and app runtime. The Android system sets up the sandbox and enforces security between apps by assigning a unique user ID (UID) to each app and by running it as that user in a separate process. Running apps within a sandbox environment ensures that any memory corruption error will only allow arbitrary code execution in the context of that particular app and with the permissions established by the OS. User-specific privileges also ensure that files created by one app cannot be read or altered by another app.

Secure IPC: Android not only supports traditional mechanisms such as filesystem, sockets and signals but also implements newer and more secure mechanisms such as Binder and Intents.

Access Control Using Manifest: Finally, Android controls app access to resources by designating certain APIs (such as camera, location, bluetooth etc.) as protected [3]. To use these resources an app needs to define its requirements in its manifest (a control file provided by every app). The user can either grant all of the requested permissions as a block or not install the app at all.

6 Architectural Design

The design of ipShield is guided by four objectives – better monitoring of sensor access, meaningful privacy abstraction, privacy rule recommendation and fine-grained control over shared data. The architectural requirements to achieve the above functionalities are

shown in Fig. 3 and can be broken down into four major blocks – (i) Databases (ii) Context Engine (iii) FirewallManager (iv) Rule-Based Obfuscator. We describe each of the blocks and their components in detail below.

Databases: We maintain two databases: Sensor Counters and Inference DB. Currently, apps have unrestricted access to the class of innocuous sensors. One of our goals in ipShield is to instrument the OS to monitor the number of sensors accessed by an app. The information is populated in the Sensor Counters database (marked (F)) and is provided as an input to the FirewallManager block. The database needs to be updated when a new sensor is accessed by an installed app or when an app in uninstalled.

Motivated by the database of virus signatures maintained by antivirus software, we maintain a similar database for mapping the list of inference categories (and their labels) that could be predicted using a combination of sensors, together with the prediction accuracy and the machine learning algorithm employed (Table 2 shows a small subset of the Inference DB). Advances in sensing coupled with increases in the sophistication of learning algorithms result in newer inference categories and improved accuracy. The inference DB (marked (E)) thus needs to be kept updated.

Context Engine: For granularity of rules, ipShield allows trusted Context Engines (marked (D)) to register and provide as input context labels. A context engine is a set of machine learning algorithms, which take as input raw sensor data and output the current context label. An inference label is same as the context label but it is inferred from the shared data by the adversary. A user can configure privacy rules to trigger on context labels.

FirewallManager interacts with the user and is responsible for generating the privacy rules. There are four different sub-blocks within FirewallManager (marked (G) through (J)).

The Semantic Firewall Configurator ((G)) takes as input the sensors accessed by an app and queries the Inference DB to present the user with a list of possible inference categories that can be predicted by the app. Using inferences instead of sensors allow us to better communicate the privacy risks to the user [52]. The user then configures a whitelist and a disjoint blacklist from the enumerated list of inference categories.

The Rule Recommender ((H)) (Section 6.2) takes as input the privacy preferences of the user expressed in terms of the whitelist and blacklist and translates them to actual privacy actions on the sensors. We observe that the privacy actions are dependent on the inference labels, the learning algorithm employed and the features used. Therefore, to keep the recommender simple and generic we limit the auto-generated privacy actions to Normal and Suppress (Section 6.1). While the auto-generated rules are binary and conservative, we provide the user with the flexibility to override them.

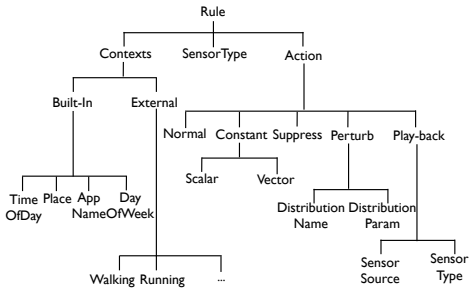

Figure 4: Tree showing all the possible options currently implemented in ipShield for constructing privacy rules. The leaf nodes of the tree are instantiated to form the privacy rules.

The Direct Firewall Configurator ((I)) allows the user to manually configure fine-grained context-aware rules. The contexts used can either be ones provided by ipShield, or ones which are externally obtained from the trusted Context Engine. ipShield is designed to operate in the Semantic mode. The Direct Configurator is an optional mode, which provides flexibility of rule configuration at the cost of increased human interaction.

Finally, the Configurator Switcher block ((J)) allows the user to switch between the Semantic and Direct Configurator modes and configure rules.

Rule-Based Obfuscator ((B)) implements the different privacy actions. It takes as input privacy rules and sensor data and, depending on the app, applies the appropriate rules to the data before releasing them.

6.1 Taxonomy of Privacy Rules

The complete list of choices for configuring privacy rules is illustrated in Fig. 4. A rule has three basic parts: Context, SensorType, and Action. We also allow conjunction (denoted by the $\wedge$ operator) of the context labels within a rule. The general form of a rule is if $(\wedge_{i=1}^{n} Context_i)$ then apply $Action$ on $SensorType$. For example, if $((TimeOfDay \text{ in } [10am-5pm]) \wedge (Place = school) \wedge (AppName = facebook))$ then apply $Action = Suppress$ on $SensorType = gps$. As shown in Fig. 4 some of the simple contexts such as TimeOfDay, DayOfWeek, Place and AppName are built into ipShield. External contexts provided by a registered Context Engine can also be used to configure rules. SensorType refers to the sensor (e.g., accelerometer, GPS, gyroscope) on which the action is to be applied.

Excluding the default action of releasing data without any changes (Normal), ipShield currently supports four different privacy actions. The Suppress action (S-block in (B)) when applied blocks data from reaching an app and the app is unable to detect any sensor event. The Constant action (K-block in (B)) allows user to replace

actual data with a constant value. The user-specified constant can be vector or scalar valued depending on the type of sensor whose data is being replaced. The Perturb action (P-block in (B)) can be used to add noise to sensor data. The noise values can be drawn from different probability distributions, the parameters of which are input to this action. Finally, the Play-back action can be used to suppress the data from the actual sensor hardware and instead send synthetic sensor measurements from an external service to the requesting app (U-shaped datapath). The synthetic data source and sensor type are input to this action. The Play-back option can be used for generating any arbitrary transformation on the data offline.

6.2 Rule Recommender

The Rule Recommender takes as input the whitelist and the blacklist of inference categories and generates a configuration for enabling or blocking of sensors accessed by an app. The goal is to ensure that only those inference labels which form part of the whitelist are allowed and those in the blacklist are blocked.

6.2.1 Problem Formulation

Let N be the number of sensors used by an app (obtained from Sensor Counters) and $\mathbf{s} = [s_1, \ldots, s_N]$ represent the sensor state vector where $s_i \in \{0,1\}$ represents the state of the i^{th} sensor. Setting s_i to 0 indicates that the sensor is disabled and a value of 1 indicates that the sensor is enabled. We denote the set of inference categories by $\mathscr{L} = \{l_1, \ldots, l_{|\mathscr{L}|}\}$. We define a mapping $M : \{0,1\}^N \times \mathscr{L} \rightarrow [0,1]$ where $M(\psi, l) = 0$ indicates that there exists no learning algorithm which can use the data streams from the sensors which are enabled as per state vector ψ and infer a label in category l. A non-zero value of $M(\psi, l)$ correspond to the maximum accuracy among all the learning algorithms that can be used to infer category l from the data streams released as per the state vector ψ. A value of 1 indicates that l can be perfectly inferred using the enabled sensors. Note, we might use different learning algorithms to predict the same labels using different sensor state vectors. The mapping M is obtained from the Inference DB. Learning algorithms typically work on features extracted from the raw sensor data. But, our current model is agnostic to features because we are sharing the raw sensor data itself and hence every required feature can be extracted from it. The set of whitelisted categories $\mathscr{W} \subseteq \mathscr{L}$, and the set of blacklisted categories $\mathscr{B} \subseteq \mathscr{L}$, are as specified by the users such that $\mathscr{W} \cap \mathscr{B} = \emptyset$. Finally, let $p_l \in \{0, \ldots, P_{max}\}$ denote the priority level set for category l by the user such that a higher value of p_l indicates higher priority. The priority levels represent a relative gradation of risk as perceived by the user. For example, $P_{max} = 3$ could correspond to *low*, *medium* and *high* levels of perceived risks. We use the above notations to formulate the inference-privacy problem as the following constrained optimization problem

$$\max_{\psi \in 2^N} \quad \sum_{l \in \mathscr{W}} M(\psi, l) 2^{p_l} - \sum_{l \in \mathscr{B}} M(\psi, l) 2^{p_l} \quad (1)$$

$$\text{s.t.} \quad \sum_{\substack{l \in \mathscr{B} \\ p_l = P_{max}}} M(\psi, l) = 0. \quad (2)$$

The objective function in Eqn. 1 is designed to maximize the prediction accuracy of the whitelisted labels and minimize the prediction accuracy of the blacklisted labels. The priorities are exponentially scaled up to account for whitelisted labels which can be detected with low accuracy than other labels but have a higher priority. The constraint in Eqn. 2 ensures that users can force blacklisted inferences to be blocked by setting their priority to P_{max}. We note that the search space in the optimization problem shown in Eqn. 1 is constrained to the vector of elements with 0's and 1's corresponding to the enabled and blocked sensors respectively. It then follows that the search space is constrained to the vertices of a hypercube. It is also easy to show that this search space is non-convex. Moreover, the optimization function depends on the relation M on which we impose no structure or even linearity. Thus, our program is non-linear integer program which is non-convex and NP-complete. We observe from our investigation of prior work and apps from Google Play (Section 8) that $N \leq 6$ for almost all the apps. Therefore, to solve a specific instance of the optimization problem above (a given choice of whitelist, blacklist, N, and priorities) we apply brute force and enumerate all possible state vector combinations. We filter out all state vectors which satisfy the blacklisted constraint and maximize the objective function over this reduced space. The output vector ψ shows which sensors should be enabled or disabled mapping preferences on inferences to privacy actions on sensors. There will be scalability issues for large N (> 15) but in practice we do not think there will be a single inference made using 15 different sensor types on a phone in the near future.

6.2.2 Numerical Example

We return to the motivating example (Section 2.1) and express it in terms of the notation described above. Thus, $\mathscr{L} = \{\text{Transportation Mode}, \text{Location}, \text{Onscreen Taps}\}$, $\mathscr{W} = \{\text{Transportation Mode}\}$ and $\mathscr{B} = \{\text{Location}, \text{OnscreenTaps}\}$. The mapping M is presented in Table 2 (under Inference Categories). We set the maximum priority level $P_{max} = 10$ throughout this example and represent a user specified priority vector as a tuple $(p_{transport}, p_{location}, p_{tap})$. We apply the algorithm above for different choices of priority vectors and report the evaluation results also in Table 2 (under column titled Evaluation).

Sensor Combination	Inference Categories			Evaluation		
	Transportation Mode	Location	Onscreen Taps	Priority1 $\{10,4,10\}$	Priority2 $\{10,0,7\}$	Priority3 $\{5,9,9\}$
GPS+ Acc + Gyro	95%	97%	80%	0	**869.4**	-875.8
GPS+WiFi	83.1%	97%	0%	**835.4**	849.9	-470.0
GPS+GSM	81.7%	98.2%	0%	820.9	835.6	-476.6
GSM+WiFi	72.9%	94.03%	0%	731.45	745.5	**-458.1**
GSM+Wifi+Acc+Gyro	92%	94.03%	80%	0	838.7	-861.6
Wifi+Acc+Gyro	91.1%	23.08%	80%	0	830.2	-498.6
GSM+Acc+Gyro	88.1%	94.03%	80%	0	798.8	-862.8
GPS	75.8%	97%	0%	760.7	775.2	-472.4
GSM	61.8%	94.03%	0%	617.8	631.9	-461.7
Acc+Gyro	84.6%	23.08%	80%	0	763.7	-500.7

Table 2: Left: A portion of the Inference DB (mapping M). Each entry (in %)is the maximum prediction accuracy for the inference category using the sensor combination. Right: The objective function (Eqn. 1) evaluated for different priority vectors and $P_{max} = 10$.

Consider $Priority1 = (10,4,10)$ as the selected priority vector. The user is not too concerned about revealing his Location and sets $p_{location}$ to 4. She however wants to strictly suppress the detection of Onscreen Taps and sets p_{tap} to 10. A high priority is also given to the whitelisted inference category by setting $p_{transport} = 10$. The objective function values for the different sensor combinations is shown in the column under heading Priority1. The maximum occurs for the combination corresponding to GPS+WiFi and is selected by the recommender. The selected sensor state vector is such that accelerometer data is suppressed in order to guarantee no leakage of the Onscreen Taps information. We also note that GPS+WiFi configuration provides higher accuracy in predicting the Transportation Mode and lower accuracy for Location prediction compared to other sensor combinations.

We consider another scenario with priority vector $Priority2 = (10,0,7)$. In this case, the user does not worry about Location disclosure, but wants to increase the prediction accuracy of the Transportation Mode while blocking the Onscreen Taps if possible. The objective function values are shown under column Priority2. The recommender selects the GPS+Accelerometer combination which is biased towards performance. In addition to meeting blacklist requirements the combination also provides the best accuracy.

Finally, the third user has high levels of concern about revealing both Location and Onscreen Taps information. She would like to trade the performance with privacy and thus selects a priority vector $Priority3 = (5,9,9)$. The resulting sensor combination chosen is GSM+WiFi. The rule recommender starts by suppressing the accelerometer data (to prevent tap inference). It then selects the combination which results in the worst Location inference from among the remaining set of combinations (GSM and GSM+WiFi), while simultaneously maximizing the whitelist accuracy.

Model-Based Augmentation of Rule Recommender: Prior research has shown that a user's various context labels and transitions between them can be captured by a Markov chain [30], by using a Dynamic Bayesian Network [47], or explicitly enumerated [20]. A user specifies a whitelist and a blacklist of inference categories, and depending on the current context label and the learned model the system can determine whether to release a context with a particular probability. In other words, the probability of release of a context should not increase the adversarial accuracy of predicting a blacklisted inference label. We envision that such model-based techniques can also be included in our recommendation system for generating a richer set of privacy rules.

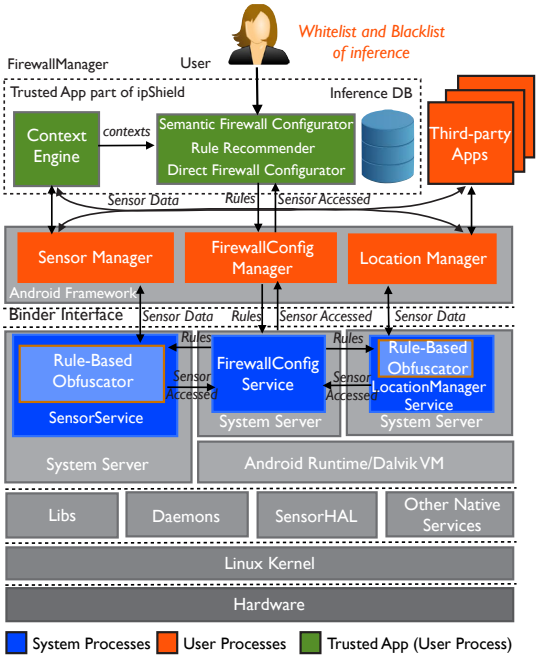

Figure 5: Implementation of ipShield on Android.

7 Implementation

The implementation of ipShield within the Android stack (Fig. 5) is described below.

7.1 Trust Model

We assume that the user installed third-party apps (e.g., from Google Play) are untrusted but do not collude with each other and share information. We trust the Linux kernel on which Android OS is built and also the Application Sandbox implemented by the kernel (Section 5). We extend the chain of trust to include the OS libraries and system services which run within the Application Sandbox and are protected by UID and group ID privileges. However, recent successful exploits from Facebook on modifying the internal data structures of the Dalvik VM [11] leads us to not trust the Application Framework components which run within the same process as the Dalvik VM.

7.2 Intercepting Data: Possible Choices

As indicated by the markers (①-⑩) in Fig. 2, there exist different operating points in both the data paths (Path-S and Path-G) at which we can intercept the sensor data, apply privacy actions, and obfuscate it. However, also associated with an operating point is the implementation complexity of the Rule-Based Obfuscator block (Ⓑ in Fig. 3) at that point and also its vulnerability to security attacks. We discuss below the trade offs in selecting an operating point.

Points ① and ⑥, correspond to modifying the kernel drivers to obfuscate data. While the drivers are protected by kernel security mechanisms, they require our implementation to be vendor specific. It is also hard to push app and rule information to the drivers and periodically update rules inside a driver.

Points ② and ⑦ correspond to changes in the SensorHAL layer. The HAL provides the abstraction between device specific kernel drivers and the Android system above. However, changing the HAL like the kernel driver has a high implementation complexity in terms of pushing app and rule information.

Points ③ and ⑧, correspond to modifying the Android system services, namely `SensorService` and `LocationManagerService` which are responsible for handling the different sensors and the GPS respectively. These services as shown in Fig. 2 run in a process separate from the app and hence are protected by the Application Sandbox. Both `SensorService` and `LocationManagerService` maintain information about installed apps, and can be easily signaled using binder calls and as we show later in Section 8 they incur low overhead while updating rules.

Points ④ and ⑨ correspond to changing the `SensorManager` and `LocationManager`, respectively. These points have the least implementation complexity, however both `SensorManager` and `LocationManager` run within the same process as the app, and hence they are not protected by process-level isolation. Recent exploits have used the above security vulnerability to modify code data structures at runtime [11].

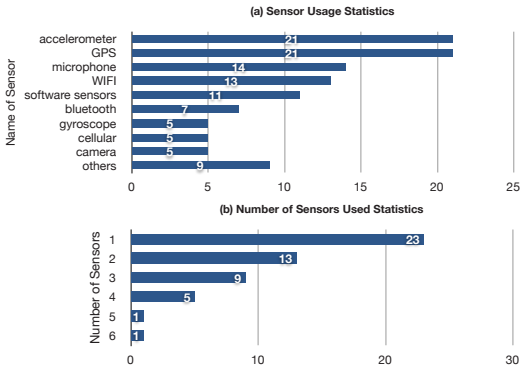

Figure 6: Statistics of sensor usage from the Inference DB.

Finally, points ⑤ and ⑩ correspond to static analysis of the app code to understand privacy violations [32]. However, the information flow approaches are often conservative, incur large instrumentation and runtime overhead, and typically stop at identification of a malicious app. Based on the available choices we decided to implement the Rule-Based Obfuscator block within the OS in the `SensorService` and `LocationManagerService` blocks in the respective data paths.

7.3 ipShield Code Blocks

ipShield is an open source project. The code for each of the blocks together with complete instructions for downloading and installing ipShield are available at [5].

7.3.1 Databases

Sensor Counters: This database, implemented as a file, maintains a counter for each sensor on a per-app basis. The counter for a sensor represents the number of events from the sensor that have been sent to the app. We use an `unsigned` $64-bit$ `long int` for our counter. Even at the maximum sampling rate of a sensor, under continuous sensing, the counter will not overflow within the lifetime of a phone. The entry for an app together with the counters are deleted when the app is uninstalled from the phone. A counter value of zero indicates that the sensor is not being used by the app. These counters are maintained by {`Sensor`, `LocationManager`}Service and are periodically written to the `/data/sensor-counter` file every minute. The permissions on the file are such that it can be read by any app but can be written to only by system services.

Inference DB: A knowledge repository generated from a survey of 60+ papers published in relevant conferences and journals over the past $3-5$ years. This

Inference Category	Labels
Transportation Mode [53]	still, walking, motorized
Device Placement [48]	hand, ear, pocket, bag
Onscreen Taps [43]	location of taps on screen
Location [19] [35] [46]	home, work, public, restaurant...
Emotion [50] [22]	happy, sad, fear, anger, neutral
Speaker [39] [46]	male/female, identity
Text Entered on Phone [43]	alphabets
Stress [40] [22]	stressful or not

Table 3: Selected inference categories from Inference DB.

database captures a wide variety of inference categories a small set of which is shown in Table 3. For each inference category, we store the prediction accuracy over the constituents labels for a particular sensor combination. If there are multiple papers using the same sensor combinations predicting the same set of labels we store details of the one with highest accuracy. We also maintain information about the set of sensors used, the features extracted from the sensor data, the classifiers used, and finally the paper title under which the results were published. In Fig. 6, we show statistics of sensor usage computed using the inference database. Based on our survey, we found that (a) GPS and accelerometer sensors are the most commonly used; (b) the number of sensors accessed by any app is almost always less than 6 (we do not include papers which use external body worn sensors in the plot, but even externally worn sensors are less than 6 types). While newer inferences are being made, we do not expect the database to change rapidly. To enable crowdsourcing of the inference DB, we have designed and published a web interface where people can contribute entries [5]. Currently, we rely on manual screening of the received entries before adding them to the Inference DB.

7.3.2 Context Engine

To allow fine-grained context-aware rules, ipShield allows trusted external context engines to register contexts that they can provide using the interface in Fig. 7(e). The user can then configure rules which will be triggered on a particular context. ipShield expects the context engines to use Android supported intents (`action`=label) as the IPC mechanism for providing the context labels.

Contexts such as battery status, contact list, ringer status etc., do not require access to sensor data and can be obtained through APIs provided by the Android OS. However, for contexts that require sensor data, the external context engine must have access to raw sensor data. To implement this when a data buffer from the HAL is received by the `SensorService` and/or the `LocationManagerService` it is first sent to the context engine to get the current context label. On receiving the context, the associated rules are then loaded and used by the Rule-Based Obfuscator to obfuscate the data buffer.

We modified the Transportation Mode app [53] to implement an activity context engine and test its integration with ipShield. In our implementation, the context engine used `SensorManager` for subscribing to accelerometer data at the rate of *SENSOR_DELAY_GAME*. This resulted in sensor data at a rate of $50Hz$ or a sample every $0.02s$. We used data buffered over a sliding window of $1s$ for inferring the activity context. On an average, the engine took about $8ms$ to generate activity context from a $1s$ accelerometer window. Even with additional overhead due to binder call and rule loading, we found that the associated rules can take effect before the next sensor data sample. This meant that our buffer size could be equal to $1s$ of data without losing any sample. In general, for keeping the buffer size bounded we observe that the processing time of the context engine together with the rule update time should be less than the inter arrival time between two data samples.

7.3.3 FirewallManager

The FirewallManager is a trusted Android app which has three different components described below.

Semantic Firewall Configurator: This is an Android activity. It reads the Sensor Counters for the installed apps and queries the inference DB for possible inference categories for each app. When launched it displays this information (Fig. 7 (a)) for the user. Once the user selects an app she is presented with the inference categories with an option to classify each into a whitelist or a blacklist (Fig. 7 (b)). The Configurator then passes the data user preferences to the Rule Recommender.

Rule Recommender: The algorithmic aspects of the rule recommender are described in detail in Section 6.2. It is implemented within the Semantic Firewall Configurator. It then uses the `FirewallConfigManager` to write the rules to `/data/firewall-config` file and also use a binder call to signal the `SensorService` and `LocationManagerService` to reload the new rules.

Direct Firewall Configurator: In this mode the user can configure context-aware privacy rules (Fig. 7 (c) and (d)). The user can specify actions on sensors used by apps, and for each action also associate either built-in contexts such as TimeOfDay, DayOfWeek, Place, or external contexts as triggers. For defining the Place context, the user can drop a marker on the map as shown in Fig. 7 (f) to annotate a $< latitude, longitude >$ tuple with a significant place name. For external contexts the Configurator implements a `BroadcastReceiver` which listens for intents. When an intent containing a particular label is received, the `BroadcastReceiver` invokes a rule loader service which passes a pre-configured set of rules associated with the label to the `FirewallConfigManager`. The `FirewallConfigManager` in turn writes the rules into the `/data/firewall-config` file and signals

both SensorService and LocationManagerService to reload the new rules. Note that the user can also explicitly request for loading a new set of rules.

7.3.4 Rule-Based Obfuscator

The Rule-Based Obfuscator block is responsible for enforcing the actions specified by the privacy rules described in Section 6.1. This block is implemented both within LocationManagerService and SensorService with the same functionality. The rules are read from the /data/firewall-config file and inserted into a HashMap for faster access. The serialization and deserialization of both rules and Sensor Counter is implemented using Google Protocol Buffer [8]. The hash for each rule is computed on the fields {appName, UID, sensorType, ruleSeqNum} where ruleSeqNum is a sequence number assigned to a rule for a sensorType. This allows multiple rules for a sensor implementing the OR operation on contexts (AND operation is implemented by allowing multiple contexts for each rule). UID is assigned to an app by Android at install time.

FirewallConfigManager, FirewallConfigService: The FirewallConfigManager interfaces with both the Semantic and the Direct Configurator modules of FirewallManager app and communicates the privacy rules to the FirewallConfigService through the binder interface. The service runs within a system process and writes the rules to the /data/firewall-config file and signals the LocationManagerService as well as the SensorService to reload the new rules.

8 Evaluation

We implemented ipShield by modifying the Android Open Source Project [2] (AOSP, branch 4.2.2_r1). We deployed and performed all our tests on the Google Nexus 4 phone (1.5GHz quad-core Qualcomm Snapdragon™ Pro, 2GB RAM).

8.1 Performance Overhead

We measured the overhead incurred by running ipShield to highlight that it is feasible to deploy it on current mobile platforms without impacting user experience in terms of battery life and app responsiveness.

8.1.1 Rule Access

Android supports four different sampling rates. On the Nexus 4 we found that on average SENSOR_DELAY_NORMAL and SENSOR_DELAY_UI are less than 10Hz, SENSOR_DELAY_GAME is around 50Hz and SENSOR_DELAY_FASTEST is around 200Hz. In Fig. 8 (a), the blue bars show the times taken to load the rules from the file (/data/firewall-config) into the

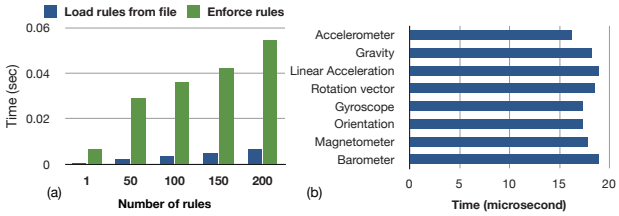

Figure 8: (a) Time taken in for the rules to load into memory and take effect. (b) Time overhead to fetch one sensor data sample sampled at SENSOR_DELAY_FASTEST.

HashMap, which are negligible. The green bars in the figure represent total time for the rules to take effect after configuration. For SENSOR_DELAY_NORMAL and SENSOR_DELAY_UI no data sample will be released before the new rules take effect even for 200 rules. In reality, we believe that the number of privacy rules will typically be less than 50, therefore for SENSOR_DELAY_GAME and SENSOR_DELAY_FASTEST less than 2 and 6 samples will be released before the 50 rules take effect, respectively.

8.1.2 Sensor Data Access

The overhead i.e., difference in time for fetching one data sample using ipShield compared to that on unmodified AOSP is shown in Fig. 8 (b). The overhead is computed by taking the average of fetching 30000 samples. Each sensor is sampled at SENSOR_DELAY_FASTEST (200Hz). The time for ipShield is averaged over the time for performing each of Constant, Perturb, and Normal (no change) actions on every accessed sample. We can see that the access overhead per sample is less than 20μsec – negligible even for the fastest sampling rate.

8.1.3 CPU and Memory Overhead

The Rule-Based Obfuscator block is part of both SensorService and LocationManagerService which run as threads inside the system_server process. For each data sample, the Rule-Based Obfuscator block is called to apply the privacy actions. We compare the overhead of the Obfuscator block with AOSP by profiling the average CPU utilization of the phone while running the Ambulation app [53] which continuously requests sensor data (GPS, accelerometer) at a rate of 1Hz on a Nexus 4 phone. CPU utilization with AOSP averaged 2%. CPU utilization with various privacy actions averaged 2.5% and never exceeded 3%. It should be noted that the CPU utilization (and hence energy consumption) will scale with the sampling rate. As shown in the energy analysis that follows this section, we believe that the overhead of ipShield is small enough to have negligible effects on overall system performance.

Memory overhead for the transformations is shown in Fig. 9 (a). The highest overhead is for Perturb and is less than 0.5MB. There is a dip in memory usage for Sup-

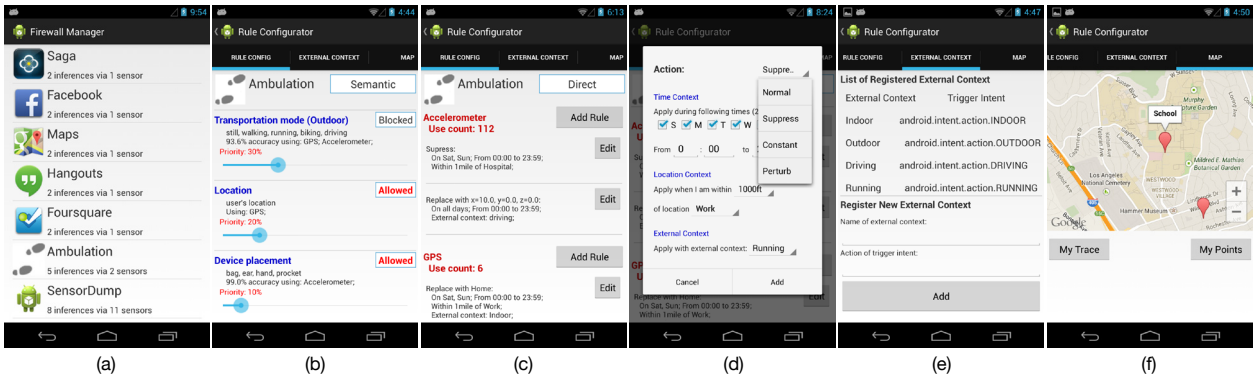

Figure 7: (a) List of installed apps showing number of sensors and number of possible inferences. (b) Semantic Firewall Configurator showing list of inference categories with option to block or allow. (c) List of rules configured for different sensors. Multiple rules with combination of contexts can be configured for each sensor. (d) Direct Firewall Configurator for privacy actions and their parameters. (e) List of external contexts registered with FirewallManager and ability to add new ones. (f) Screen to annotate significant places on the map (provides built-in Location context for rules).

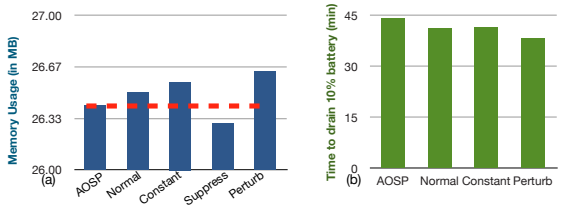

Figure 9: (a) Memory Overhead. (b) Energy Overhead.

press action which lowers the average memory overhead over all the operations in ipShield to around 0.07MB.

8.1.4 Energy Overhead

We compare the energy overhead of ipShield to AOSP by plotting the time to drain the phone battery from 100% to 90%, while the Ambulation app is continually running in the foreground for Transportation Mode inference. All network interfaces and radios are turned off, and the screen display is on at the lowest brightness. We acquire a CPU wakelock in the app to prevent the phone from sleeping. The inference frequency of the app is set to 4Hz. We measure the drain due to three actions: Normal, Constant, and Perturb which will consume more power than the AOSP. Fig. 9 (b) shows the results: ipShield on average drains the battery 3min 37s ($\sim 8.2\%$) faster than AOSP, which we consider as a marginal overhead. In typical usage scenarios where the screen is at a higher brightness setting and the network subsystem is active we expect the energy overhead for ipShield to be relatively lower.

8.2 Vulnerability of Current Apps

We did a survey of the top 60 free apps from Google play store to find the different sensors used by these apps. We installed and executed each of the apps from the play store, and noted the permissions to sensors requested by the apps at install time, and also the sensors which were being accessed without permission using ipShield. We also made use of the description of the app provided at the app store for additional information (if any). This provided the list of sensors used by each app. We then used the Inference DB to create the association between app and possible inference categories as shown in Table 4. The results from this survey validates our claim that GPS and accelerometer which are the most used sensors in academic research (Fig. 6) are also the most widely used sensors in apps. We further note, that many of these apps have access to data from innocuous sensors, combinations of which can be maliciously used to predict a lot more inferences than what they advertise.

8.3 Case Studies: Revisited

We now illustrate how ipShield can be used to configure simple rules to overcome the privacy issues outlined in the examples in Section 2.

***Transportation Mode and KeyLogging*:** While suppressing accelerometer at all times is a naive solution, to obtain better utility from the app, the user can use the Direct Rule Configurator to select an external context KEYBOARD_UP, and use it to define the following rules: If $((TimeOfDay \text{ in } [12am - 11:59pm]) \wedge (ExternalContext = KEYBOARD\_UP) \wedge (AppName = Ambulation))$ then apply $action = Suppress$ on $SensorType = acc$; and a similar rule for suppressing $SensorType = gyro$. We exploit the fact that it is sufficient to block the accelerometer and gyroscope data while the softkeyboard is active to protect against keylogging. On executing the above rules, the act of suppression will inform an

Sensor Combination	App Name	Inference Categories
GPS	Twitter, iHeartRadio, Calorie Counter, Amazon, eBay, MyTracks, Google Earth (and 12 more..)	A1: Loc, Speed, Route
Acc	Despicable me, Subway Surfers, Accupedo Pedometer	A2: TM, Device Placement, Text on Keyboard
Audio	Snapchat, Vine, Google Translate, Cadiograph	A3: Speaker, Loc, Emotions, Stress
Acc + GPS	Picsart, Noom Weight Loss Coach, Temple Run	A1 + A2
GPS + Audio	FB, FB Messenger, Tango, Whatsapp, Shazam, GoSMS	A1 + A3
Acc + GPS + Audio	Instagram, Neon motocross	A1 + A2 + A3
Acc + Pro + GPS + Audio	Skype	A1 + A2 + A3
Acc + Rot + GPS	Maps	A1 + A2, Onscreen Taps, Text Entered on Phone
Acc + Gyro + Pre + GPS	Saga Lifelogging	A1 + A2, Onscreen Taps, Text Entered on Phone

Table 4: Sensors and possible inferences from top apps in Google Play Store (all have access to network). Loc:Location, Acc:acclerometer, Pro:proximity, Rot:rotation vector, Gyro:gyroscope, Pre:pressure, TM:Transportation Mode.

adversary that the user is entering text, but she cannot infer anything more. The Ambulation app will now continue to work at all times when the user is not entering text, maximizing its utility to the user.

***Saga and Location*:** A user would often like to keep some of his visits to sensitive places private. ipShield allows the user to configure the following rules to spoof her location trace: (1) If $((TimeOfDay$ in $[12am-11:59pm]) \wedge (Place = Friend'sHome) \wedge (AppName = Saga))$ then apply $action = Constant$ and $value = Starbucks$ on $SensorType = GPS$; (2) If $((TimeOfDay$ in $[12am-11:59pm]) \wedge (Place = Bar) \wedge (AppName = Saga))$ then apply $action = Constant$ and $value = Restaurant$ on $SensorType = GPS$; As we mentioned earlier user can configure labels such as *Starbucks, friend's home, bar* using the map interface in ipShield. To ensure plausibility of the shared location data the perturbation performed, or even the constant value provided, should conform to a map [37].

9 Concluding Remarks

While phones have evolved into sophisticated sensing platforms the corresponding sensing stack where starting at the raw sensor data meaningful data abstractions are created at each layer (akin to a communication stack) [26] has not yet taken shape. Efforts like CondOS [23], together with architectural changes such as dedicated co-processors for context detection [4] are steps in the direction towards introducing greater semantic clarity for shared data. With such a stack in place it is then a natural design choice to have a privacy system built within the OS itself exploiting the semantic granularity of data for improved privacy.

With ipShield we advocated the above design philosophy and took the first step towards creating a framework with architectural changes built within the Android OS to protect user privacy. We introduced better monitoring of accessed resources, proposed a user-understandable privacy abstraction in the form of possible inferences, allowed users to configure semantic privacy rules, and ensured that user preferences are securely enforced.

Orthogonal to the enforcement of rules is their creation. In future, to minimize user interaction in rule formulation it is imperative that systems are able to learn rules based on the semantic similarity of shared data and basic user preferences. With respect to granularity of rules, even with user participation privacy rules can often tend to become conservative impacting the app utility. To this end, careful integration of ipShield with various static analysis tools [32] could provide better insight into the working of apps and in the creation of balanced rules.

The other pertinent question is regarding the selection of a suitable set of privacy actions. Integration of cryptographic solutions would enrich the spectrum of available actions. In addition, currently ipShield does not handle traditional side-channels attacks and it will be an interesting extension to the current system.

Finally, any such system should be able to run on resource constrained platforms. Our experiments with ipShield indicate that it has low performance overhead and can run continuously on various mobile platforms without impacting app responsiveness.

Acknowledgement

We thank our shepherd Jonathan M. Smith, the anonymous reviewers, and our collaborators Haksoo Choi, Nicolas Bitouze and Lara Dolecek for their valuable comments. This work was supported in part by the U.S. ARL, U.K. Ministry of defense (MoD) under Agreement Number W911NF-06-3-0001, by the NSF under awards CNS-0910706 and CNS-1213140, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via US Navy (USN) SPAWAR Systems Center Pacific (SSC-Pac). Any findings in this material are those of the author(s) and do not reflect the views of any of the above funding agencies. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Android 4.3's Hidden App Permission Manager. http://www.androidpolice.com/2013/07/25/app-ops-android-4-3s-hidden-app-permission-manager-control-permissions-for-individual-apps/.

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

[3] Android Security Overview. http://source.android.com/devices/tech/security/.

[4] Apple M7 Co-Processor. http://en.wikipedia.org/wiki/Apple_M7.

[5] ipShield: A Framework For Enforcing Context-Aware Privacy. http://tinyurl.com/ipshieldgit.

[6] Pausing Google Play: More Than 100,000 Android Apps May Pose Security Risks. https://www.bit9.com/download/reports/Pausing-Google-Play-October2012.pdf.

[7] PDroid patch for Android Jelly Bean. http://github.com/gsbabil/PDroid-AOSP-JellyBean.

[8] Protocol Buffers. https://developers.google.com/protocol-buffers/.

[9] RunKeeper. http://www.runkeeper.com/.

[10] Saga Lifelogging. http://www.getsaga.com/.

[11] Under the Hood: Dalvik patch for Facebook for Android. https://www.facebook.com/notes/facebook-engineering/under-the-hood-dalvik-patch-for-facebook-for-android/10151345597798920.

[12] AGARWAL, Y., AND HALL, M. Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (2013), MobiSys '13, pp. 97–110.

[13] ASONOV, D., AND AGRAWAL, R. Keyboard acoustic emanations. In *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on* (2004), pp. 3–11.

[14] AVIV, A. J., SAPP, B., BLAZE, M., AND SMITH, J. M. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference* (2012), ACSAC '12, pp. 41–50.

[15] BADEN, R., BENDER, A., SPRING, N., BHATTACHARJEE, B., AND STARIN, D. Persona: An online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (2009), SIGCOMM '09, pp. 135–146.

[16] BAO, L., AND INTILLE, S. S. Activity recognition from user-annotated acceleration data. *Pervasive LNCS 3001* (2004), 1–17.

[17] BARRERA, D., KAYACIK, H. G., VAN OORSCHOT, P. C., AND SOMAYAJI, A. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), ACM, pp. 73–84.

[18] BERESFORD, A. R., RICE, A., SKEHIN, N., AND SOHAN, R. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications* (2011), HotMobile '11, pp. 49–54.

[19] BROUWERS, N., AND WOEHRLE, M. Detecting dwelling in urban environments using gps, wifi, and geolocation measurements. In *Proc. 2nd Intl Workshop on Sensing Applications on Mobile Phones* (2011), pp. 1–5.

[20] CHAKRABORTY, S., BITOUZÉ, N., SRIVASTAVA, M., AND DOLECEK, L. Protecting data against unwanted inferences. In *Information Theory Workshop (ITW), 2013 IEEE* (2013), pp. 1–5.

[21] CHAKRABORTY, S., RAGHAVAN, K. R., JOHNSON, M. P., AND SRIVASTAVA, M. B. A framework for context-aware privacy of sensor data on mobile systems. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications* (2013), HotMobile '13, pp. 11:1–11:6.

[22] CHANG, D. F. K.-H., AND CANNY, J. Ammon: A speech analysis library for analyzing affect, stress, and mental health on mobile phones. *Proceedings of PhoneSense 2011* (2011).

[23] CHU, D., KANSAL, A., LIU, J., AND ZHAO, F. Mobile apps: it's time to move up to condos. HotOS.

[24] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation* (2010), OSDI'10, pp. 1–6.

[25] ENCK, W., ONGTANG, M., AND MCDANIEL, P. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (2009), CCS '09, pp. 235–245.

[26] ESTRIN, D., AND SIM, I. Open mHealth Architecture: An Engine for Health Care Innovation. *Science 330*, 6005 (2010), 759–760.

[27] FELT, A. P., CHIN, E., HANNA, S., SONG, D., AND WAGNER, D. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (2011), CCS '11, pp. 627–638.

[28] FELT, A. P., HA, E., EGELMAN, S., HANEY, A., CHIN, E., AND WAGNER, D. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (2012), SOUPS '12, pp. 3:1–3:14.

[29] GENKIN, D., SHAMIR, A., AND TROMER, E. Rsa key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857, 2013.

[30] GÖTZ, M., NATH, S., AND GEHRKE, J. Maskit: Privately releasing user context streams for personalized mobile applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (2012), SIGMOD '12, pp. 289–300.

[31] HAN, J., OWUSU, E., NGUYEN, L., PERRIG, A., AND ZHANG, J. Accomplice: Location inference using accelerometers on smartphones. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on* (2012), pp. 1–9.

[32] HOLAVANALLI, S., MANUEL, D., NANJUNDASWAMY, V., ROSENBERG, B., AND SHEN, F. Flow permissions for android. In *Tech Report*.

[33] HORNYACK, P., HAN, S., JUNG, J., SCHECHTER, S., AND WETHERALL, D. 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* (2011), CCS '11, pp. 639–652.

[34] JEON, J., MICINSKI, K. K., VAUGHAN, J. A., REDDY, N., ZHU, Y., FOSTER, J. S., AND MILLSTEIN, T. Dr. android and mr. hide: Fine-grained security policies on unmodified android.

[35] KIM, D. H., KIM, Y., ESTRIN, D., AND SRIVASTAVA, M. B. Sensloc: Sensing everyday places and paths using less energy. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* (2010), SenSys '10, pp. 43–56.

[36] KÖPF, B., AND BASIN, D. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (2007), CCS '07, pp. 286–296.

[37] KRUMM, J. A survey of computational location privacy. *Personal Ubiquitous Comput. 13*, 6 (Aug. 2009), 391–399.

[38] LEE, S., WONG, E. L., GOEL, D., DAHLIN, M., AND SHMATIKOV, V. πbox: A platform for privacy-preserving apps. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (2013), NSDI'13, pp. 501–514.

[39] LIU, B., JIANG, Y., SHA, F., AND GOVINDAN, R. Cloud-enabled privacy-preserving collaborative learning for mobile sensing. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems* (2012), Sensys '12, pp. 57–70.

[40] LU, H., FRAUENDORFER, D., RABBI, M., MAST, M. S., CHITTARANJAN, G. T., CAMPBELL, A. T., GATICA-PEREZ, D., AND CHOUDHURY, T. Stresssense: Detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (2012), Ubicomp '12, pp. 351–360.

[41] LU, H., PAN, W., LANE, N. D., CHOUDHURY, T., AND CAMPBELL, A. T. Soundsense: Scalable sound sensing for people-centric applications on mobile phones. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services* (2009), MobiSys '09, pp. 165–178.

[42] MARQUARDT, P., VERMA, A., CARTER, H., AND TRAYNOR, P. (sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (2011), CCS '11, pp. 551–562.

[43] MILUZZO, E., VARSHAVSKY, A., BALAKRISHNAN, S., AND CHOUDHURY, R. R. Tapprints: Your finger taps have fingerprints. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services* (2012), MobiSys '12, pp. 323–336.

[44] MUN, M., HAO, S., MISHRA, N., SHILTON, K., BURKE, J., ESTRIN, D., HANSEN, M., AND GOVINDAN, R. Personal data vaults: A locus of control for personal data streams. In *Proceedings of the 6th International COnference* (2010), Co-NEXT '10, pp. 17:1–17:12.

[45] NAUMAN, M., KHAN, S., AND ZHANG, X. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (2010), ACM, pp. 328–332.

[46] NIRJON, S., DICKERSON, R. F., ASARE, P., LI, Q., HONG, D., STANKOVIC, J. A., HU, P., SHEN, G., AND JIANG, X. Auditeur: A mobile-cloud service platform for acoustic event detection on smartphones. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (2013), MobiSys '13, pp. 403–416.

[47] PARATE, A., CHIU, M.-C., GANESAN, D., AND MARLIN, B. M. Leveraging graphical models to improve accuracy and reduce privacy risks of mobile sensing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services* (2013), MobiSys '13, pp. 83–96.

[48] PARK, J.-G., PATEL, A., CURTIS, D., TELLER, S., AND LEDLIE, J. Online pose classification and walking speed estimation using handheld devices. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (2012), UbiComp '12, pp. 113–122.

[49] PLARRE, K., RAIJ, A., HOSSAIN, S., ALI, A., NAKAJIMA, M., AL'ABSI, M., ERTIN, E., KAMARCK, T., KUMAR, S., SCOTT, M., SIEWIOREK, D., SMAILAGIC, A., AND WITTMERS, L. Continuous inference of psychological stress from sensory measurements collected in the natural environment. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on* (2011), pp. 97–108.

[50] RACHURI, K. K., MUSOLESI, M., MASCOLO, C., RENTFROW, P. J., LONGWORTH, C., AND AUCINAS, A. Emotionsense: a mobile phones based adaptive platform for experimental social psychology research. In *Proceedings of the 12th ACM international conference on Ubiquitous computing* (2010), pp. 281–290.

[51] RAHMAN, M. M., ALI, A. A., PLARRE, K., AL'ABSI, M., ERTIN, E., AND KUMAR, S. mconverse: Inferring conversation episodes from respiratory measurements collected in the field. In *Proceedings of the 2Nd Conference on Wireless Health* (2011), WH '11, pp. 10:1–10:10.

[52] RAIJ, A., GHOSH, A., KUMAR, S., AND SRIVASTAVA, M. Privacy risks emerging from the adoption of innocuous wearable sensors in the mobile environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011), CHI '11, pp. 11–20.

[53] REDDY, S., MUN, M., BURKE, J., ESTRIN, D., AND HANSEN, MARK A ND SRIVASTAVA, M. Using mobile phones to determine transportation modes. *ACM Trans. Sen. Netw. 6*, 2 (Mar. 2010), 13:1–13:27.

[54] SONG, D. X., WAGNER, D., AND TIAN, X. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10* (2001), SSYM'01, pp. 25–25.

[55] THURM, S., AND KANE, Y. Your apps are watching you. The Wall Street Journal, 2012.

[56] TOOTOONCHIAN, A., SAROIU, S., GANJALI, Y., AND WOLMAN, A. Lockr: Better privacy for social networks. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies* (2009), CoNEXT '09, pp. 169–180.

[57] VUAGNOUX, M., AND PASINI, S. Compromising electromagnetic emanations of wired and wireless keyboards. In *Proceedings of the 18th Conference on USENIX Security Symposium* (2009), SSYM'09, pp. 1–16.

[58] ZHAN, A., CHANG, M., CHEN, Y., AND TERZIS, A. Accurate caloric expenditure of bicyclists using cellphones. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems* (2012), SenSys '12, pp. 71–84.