ARMOUR US: Android Runtime Zero-permission Sensor Usage Monitoring from User Space

Yan Long*
Northeastern University
Boston, Massachusetts, USA
y.long@northeastern.edu

Jiancong Cui*
Northeastern University
Boston, Massachusetts, USA
cui.jianc@northeastern.edu

Yuqing Yang
The Ohio State University
Columbus, Ohio, USA
yang.5656@osu.edu

Tobias Alam
University of Michigan
Ann Arbor, Michigan, USA
tobiasal@umich.edu

Zhiqiang Lin
The Ohio State University
Columbus, Ohio, USA
zlin@cse.ohio-state.edu

Kevin Fu
Northeastern University
Boston, Massachusetts, USA
k.fu@northeastern.edu

Abstract

This work investigates how to monitor access to Android zero-permission sensors which could cause privacy leakage to users. Moreover, monitoring such sensitive access allows security researchers to characterize potential sensor abuse patterns. Zero-permission sensors such as accelerometers have become an indispensable part of Android devices. The critical information they provide has attracted extensive research investigating how data collectors could capture more sensor data to enable both benign and exploitative applications. In contrast, little work has explored how to enable data providers, such as end users, to understand sensor usage. While existing methods such as static analysis and hooking-based dynamic analysis face challenges of requiring complicated development chains, rooting privilege, and app-specific reverse engineering analysis, our work aims to bridge this gap by developing ARMOUR for user-space runtime monitoring, leveraging the intrinsic sampling rate variation and convergence behaviors of Android. ARMOUR enables privacy-aware users to easily monitor how third-party apps use sensor data and support security researchers to perform rapid app-agnostic sensor access analysis. Our evaluation with 1,448 commercial applications shows the effectiveness of ARMOUR in detecting sensor usage in obfuscated code and other conditions, and observes salient sensor abuse patterns such as 50% of apps from seemingly sensor-independent categories accessing data of multiple zero-permission sensors. We analyze the impact of Android's recent policy changes on zero-permission sensors and remaining technical and regulatory problems.

CCS Concepts

• **Security and privacy → Mobile and wireless security**; **Malware and its mitigation**.

Keywords

Android, zero-permission, sensors, runtime monitoring, privacy

*Co-first authors with equal contributions.

WiSec 2025, Arlington, VA, USA

ACM ISBN 979-8-4007-1530-3/2025/06
https://doi.org/10.1145/3734477.3734704

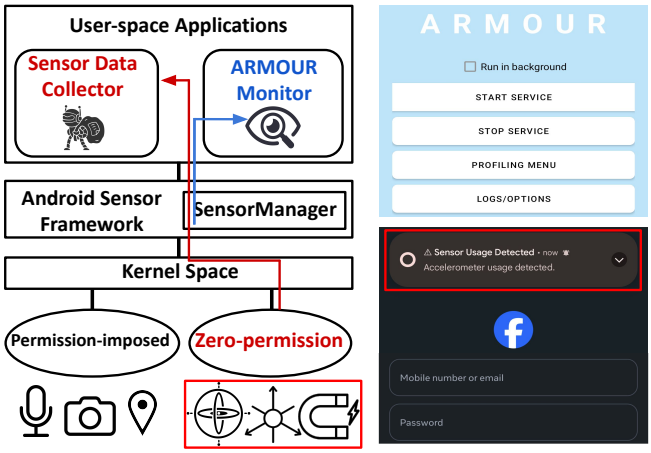

Figure 1: Although third-party apps can collect zero-permission sensor data almost without regulations in existing Android ecosystems, ARMOUR provides a complementary defender capability of monitoring sensor access information solely from the user space.

ACM Reference Format:
Yan Long, Jiancong Cui, Yuqing Yang, Tobias Alam, Zhiqiang Lin, and Kevin Fu. 2025. ARMOUR US: Android Runtime Zero-permission Sensor Usage Monitoring from User Space. In *18th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2025), June 30-July 3, 2025, Arlington, VA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3734477.3734704

1 Introduction

Sensors in smartphones and wearable devices have become a cornerstone of the seamless interactions between the physical world and cyberspace. As of 2025, there are over 6.5 billion smartphone users worldwide with Android-powered devices taking up over 80% of the market share [19]. Android devices are providing an increasing number of sensor hardware with growing capabilities to support software in acquiring more information from the physical world, enabling a wide range of applications such as health monitoring and activity recognition [30, 36, 38, 44]. While Android allows third-party apps to access and analyze sensor data for diverse purposes, the current Android platform provides very limited

mechanisms for monitoring and revealing third-party applications' sensor usage. In particular, applications can access readings of accelerometers, gyroscopes, magnetometers, and other types of sensors which can capture important motion and environmental information of the physical devices and users, without requiring user permissions or even notifying users of their usage. Such sensors are thus referred to as *zero-permission sensors* (Figure 1).

The unmonitored usage of zero-permission sensors creates nearly unregulated information channels that may leak sensitive information. This risk has manifested in a significant number of recent research works that demonstrate how zero-permission sensor data could contain critical private information. For example, smartphones' accelerometer and gyroscope readings contain vibration signals caused by human body movements and even sound waves of voice, which further enable the inference of users' age, identity, speech, location, and password inputs [4, 8, 15, 21, 32, 39, 42, 48, 53, 59]. Another example is the feasibility of identifying various user-device interactions by analyzing the electromagnetic signals embedded in magnetometer readings that are generated by smartphone CPUs or displays [17, 35, 40, 43]. Research has also shown similar problems on emerging AR/VR devices, which could present more pervasive threats given the predicted continuous wearing of these devices [49, 57]. While the possible threats identified by prior research have not yet been analyzed on commercial apps, the mere fact that third-party apps have unmonitored access to information-rich sensor data has already raised significant concerns.

As a result, we observe an information asymmetry gap between data collectors (e.g., application developers) and data providers (e.g., application users): *Despite extensive research investigating how to collect more sensor data to enable various applications, little work has explored how to meet the needs of researchers and data providers to monitor and analyze the sensor usage.* Although there exist commercial privacy-preserving applications [2, 3] that support users in tracking the usage of permission-imposed cameras and microphones, none of them is capable of monitoring zero-permission sensors. This overlooked fundamental capability calls for the development of mechanisms that allow end users and researchers to understand the pattern of zero-permission sensor access. Several existing techniques may facilitate such analysis, but still suffer from limitations. On one hand, *Static Analysis* methods may be used to detect codes accessing sensors but face challenges in three aspects: (1) lack of support for user-side runtime monitoring, (2) challenges in creating time-variant information channel, and (3) lack of semantic information among obfuscated and native codes. On the other hand, despite that *Hooking-based Dynamic Analysis* methods may address these challenges to some degree, they still face three limitations: (1) excessive expertise is required to use sophisticated software development chains, (2) physical devices with escalated privileges such as rooting are needed to perform monitoring, and (3) prior knowledge of and customization for each application needs to be obtained prior to analyzing them.

Our work addresses these constraints and contributes complementary capabilities by developing a methodology and tool for *accessible user-space runtime monitoring* of Android zero-permission sensor usage. Our tool ARMOUR[1] is capable of detecting the time of access, type of sensor, and the sampling rates accessed by third-party applications. ARMOUR is a privilege-free, off-the-shelf Android module that can run as an individual background monitoring app (Figure 1) or be seamlessly incorporated into other trusted privacy-preserving applications. It enables non-expert end users to easily monitor how third-party applications may be harvesting their sensor data and helps security researchers rapidly identify and characterize abuse patterns of zero-permission sensors. ARMOUR builds upon a key observation of the Android sensor framework's rule of sampling rate variation and convergence. Specifically, we observe that the actual instant sampling rates Android OS provides to different applications are interdependent, creating an information channel that can be leveraged by a trusted defender application to monitor the sensor usage of concurrent applications. ARMOUR running in the user space is thus able to detect sensor usage by other third-party applications by listening for sampling rate changes in its own sensor data packets. Our tests show that ARMOUR can detect sensor usage across diverse hardware and software platforms and is robust against native codes and obfuscations. Besides user-installed applications, it can also detect the usage of zero-permission sensors by websites. The utilization of this previously unexplored behavior of the Android sensor framework provides unique light-weight, user-space monitoring capabilities that enable users and security analysts to bridge the information asymmetry gap revealed by prior works.

Our evaluation aims to both measure the performance of ARMOUR in terms of its detection accuracy and runtime overhead, and use ARMOUR to characterize real-world sensor usage of commercial applications. In the former, we collected 50 apps known to use sensors and found that ARMOUR could detect all the usage. Additional prolonged testing of no sensor activities confirmed ARMOUR's ability to avoid false positive reports. When running continuously as a background monitoring application, ARMOUR only incurs an overhead of about 2.6% phone battery usage per hour. In the latter, we built another dataset with 1,398 of the most popular Google Play applications in 35 categories and performed in-depth analyses, resulting in multiple insights. For example, our results show that certain app categories such as Books, Finance, and Shopping have unexpectedly high sensor usage without clear motivation and justification. Sensor-based tracking and identification services commonly used by Finance applications confirm that sensor data is capable of capturing privacy-sensitive information. By comparing sensor sampling rate distributions in older and newer Android versions, we verified the positive impact of the HIGH_SAMPLING_RATE_SENSORS permission introduced in Android 12 in terms of security, but observed outstanding problems. For instance, apps may actively request the highest possible sampling rates. Moreover, device manufacturers may bypass Android's high sampling rate regulations, thus undermining the positive impact of the introduced permission. We also analyze case studies of abnormal sensor usage patterns measured by ARMOUR. For example, 3.6% of the apps continue to access sensors on certain smartphones even when their GUI is terminated by users. Manual

[1]Code and dataset available at https://github.com/longyan97/ARMOUR

testing demonstrates additional sensor usage that can be triggered by GUI interactions such as user login, indicating that our work's result provides a lower-bound measurement that future works can build upon. Overall, our measurements reveal obvious zero-permission sensor abuse problems that could motivate future research in providing better user privacy protections, refining sensor data access policies, and performing in-depth sensor usage attribution analysis. Our major contributions are summarized as:

- The problem formulation of zero-permission sensor usage monitoring from the defender standpoint and the first proposed mechanism for user-space runtime detection. The tool ARMOUR developed for end users and researchers complements existing Android OS and static/dynamic analysis capabilities, making a key step toward accessible sensor data privacy controls.
- A dedicated Android app dataset collected for zero-permission sensor usage analysis that consists of 1,448 popular apps. The open-source dataset's results provide ground-truth and baseline measurements that other works of Android sensor security and privacy can compare with.
- The detailed analysis of existing sensor abuse patterns in popular commercial Android applications. Our observations highlight abnormal and unjustified sensor data access, opening up possible research venues for improving the technical control and policy regulations of zero-permission sensor data access.

2 Background

This section provides the necessary background for understanding the current status and remaining gaps of zero-permission problems.

2.1 Android Sensor Framework

Figure 1 provides an overview of the Android sensor framework. Android provides two main categories of sensors, namely permission-imposed sensors including cameras, microphones, GPS, etc., and zero-permission sensors. When third-party applications want to access permission-imposed sensors, users will be prompted to decide whether to allow the sensor usage. Zero-permission sensors, on the other hand, can be used without requiring installation-time or run-time consent.

Zero-permission Sensors. Table 2 lists the main categories and representative instances of zero-permission sensors. Motion sensors such as accelerometers and gyroscopes can collect information about the movement of the device, which has been shown to reflect the physical activities of users such as walking, typing, etc. Position sensors such as magnetometers collect information about the device's physical positions and can also embed the electromagnetic characteristics of the smartphone's surroundings.

Risks and Existing Countermeasures. Although previously deemed non-sensitive, research in the past few years has proven that such zero-permission sensors' data actually contain significantly more critical and fine-grained information than what users and smartphone OS designers expected and can be exploited by third-party applications to identify smartphone users, steal touchscreen inputs, etc. Section 6.1 provides more details about the privacy-sensitive information zero-permission sensor data could contain.

Android made a series of adjustments to its policies to counteract these emerging threats discovered by previous research. Starting Android 9, apps are required to run in foreground services to access sensor data in the background [22]. Foreground services create visible dialogues on the menu bar to notify users that the application is running in the background. Nevertheless, there is still no way of knowing if apps are collecting sensor data. Android 12 starts to limit the highest available sampling rate to 200 Hz for common sensor usage by third-party apps, where any app that needs a higher sampling rate is required to declare the use in the manifest file through the HIGH_SAMPLING_RATE_SENSORS permission and explain the purpose of high sampling rates to pass the review of Google Play store [23, 47]. However, the high sampling rate is a normal-level permission that does not require either installation-time or run-time consent from the users [24]. Furthermore, recent research has shown that sampling rates lower than 200 Hz are still sufficient to extract a large portion of critical information that can be recognized by emerging machine learning and deep learning algorithms [8, 14]. As a result, there is still an urgent need for monitoring zero-permission sensor usage.

2.2 SensorManager Interface

The SensorManager class in Android provides the interface for application developers to interact with the hardware of zero-permission sensors. The applications need to register for a sensor event listener with a callback that processes received new sensor readings (onSensorChanged). When registering the listener, the application needs to request a desired sampling rate of the sensor. Android has four pre-defined sampling rate instances, namely SENSOR_DELAY_NORMAL/UI/GAME/FASTEST which have increasing numeric values approximately ranging from 5 Hz to 400 Hz on most smartphones. The actual value of each instance is implemented by smartphone manufacturers and varies with different types of sensors. Application developers can also directly specify the desired sampling rate values (in millisecond sampling intervals) without using these pre-defined values. Similarly, the range of supported sampling rates for each phone model and each sensor could vary.

It is important to note that the actual sampling rate of the sensor readings provided by the Android OS can be different from what the applications request because the OS also needs to balance the bandwidths of different system operations besides passing sensor readings to applications [25]. This enables the detection method of ARMOUR (Section 3.2).

3 Methodology & Design

Given the emerging problems of nearly unlimited access to zero-permission sensor data, our work proposes a user-space runtime sensor usage monitoring mechanism and design ARMOUR to support researchers and end users.

3.1 Threat & System Model

We assume third-party apps as data collectors want to acquire data from zero-permission sensors without revealing the use of these sensors and these apps can run in either the foreground or background. ARMOUR takes the defender's role in the form of a user-space trusted application that data providers such as security

researchers and end users can run continuously in the background to monitor details of other third-party apps' sensor usage. Given that ARMOUR's working principle (Section 3.2) allows it to monitor OS-wise sensor usage instead of individual app's usage, we assume that a data provider that wants to associate detected sensor usage to an exact third-party app can ensure only ARMOUR and this app are running, which can be achieved by terminating or uninstalling other background user-space apps.

3.2 Sampling-based Sensor Usage Detection

This section introduces the working principle of ARMOUR and characterizes its capability of detecting various sensor usage.

3.2.1 Instant Sampling Rate Variation & Convergence. ARMOUR detects Android sensor usage by observing variations in the actual *instant sampling rate* of sensor readings provided by Android OS, which could be different from the sampling rate requested by an application (denoted as f_{req}). The instant sampling rate at a certain system time t can be calculated as

$$f_{inst}(t) = 1/(T_a - T_b) \quad (1)$$

where T_a and T_b stand for the discrete timestamps of the current and last sensor data packet received via the SensorEvent data structure. Our empirical tests with the SensorManager interface show that starting another application could change the instant sampling rates received by a running application that is already collecting sensor data, as shown in Figure 2.

Based on this phenomenon, we hypothesize that when different applications register to access the same sensor, e.g., the accelerometer on the phone, the instant sampling rates available to each application will be *interdependent*. To verify this hypothesis, we ran several instances of a custom sensor-access application, each requesting a different sampling rate. The test results reveal a uniform sampling rate convergence rule:

> *When multiple applications access the same sensor's data, the instant sampling rate provided to them converges to the highest OS-supported rate requested among all running applications.*

This implies that a user-space defender application can probe the sensor usage of other applications by checking the instant sampling rate of its own sensor data. Specifically, the defender can register an unusually low sampling rate and observe the increase in its instant sampling rates as a sign of the monitored applications using the same sensor, as demonstrated in Figure 2. Essentially, this mechanism creates a benign covert communication channel between different applications. Our tests find this rule to be independent of when the applications initiate the sensor access requests and whether the applications are running in the background or foreground. We were able to align these observations with the official documentation of Android [45], which confirms the convergence to the maximum requested sampling rate but does not specify possible instant sampling rate variation behaviors in different background/foreground scenarios. Our work aims to provide detailed characterizations of this Android's intrinsic but unexplored behavior for building ARMOUR and collecting the first dataset for evaluating zero-permission sensor usage.

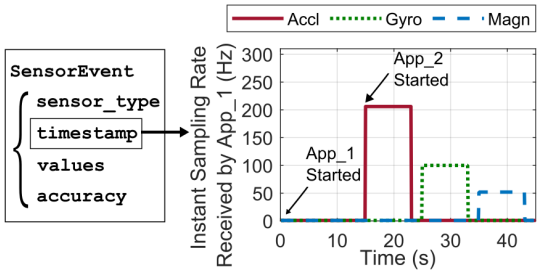

Figure 2: The change of instant sampling rates detected by an app (App_1) when another app (App_2) starts running and using sensors. ARMOUR calculates the instant sampling rates using the timestamps of the SensorEvent data packets.

3.2.2 Detection Capability. Our next tests with several self-made sensor data collection apps verify that ARMOUR could detect zero-permission sensor usage across different smartphone hardware and Android software and is immune to code obfuscations.

Device & Sampling Rate Range. ARMOUR is able to monitor sensor usage on Android devices that support the sensor management framework. This includes smartphones, smartwatches, and other Android-powered devices running Android 8 (released in 2017) and all newer versions of Android. Our tests verified the feasibility of using ARMOUR on six common smartphone models from Google, Samsung, etc., as listed in Table 3. Nevertheless, the condition of observing sampling rates higher than the requested value f_{req} suggests that ARMOUR cannot detect certain usage smaller than or equal to this value and thus have a limited range of detectable sampling rates. An informed defender will set f_{req} to the minimum supported sampling rate, denoted as f^i_{min} for sensor i, ensuring the minium sampling rate is the only usage that is not detected. Different manufacturers and phone models may differ in their implementations of the minimum supported sampling rates, as shown by Table 3 for the six phones. While the minimum sampling rate usage will inevitably be overlooked, we hypothesize that this is relatively low likelihood, and further measure to what degree this could affect the detection performance in Section 4.2 and Section 5.

Code Obfuscation & Native Codes. Code obfuscation and native code are popular techniques used by commercial app developers to prevent reverse engineering and achieve better runtime performance. Despite the significant challenges these techniques pose to static analysis methods (Section 6.2), ARMOUR is found to be immune to these techniques because of its runtime dynamic analysis nature that does not require processing any code/binary files. Our tests with a customized app obfuscated with the ProGuard obfuscator and another customized app that utilizes C/C++ native codes verified that ARMOUR could detect zero-permission sensor usage in both cases while a previous attempt of adapting existing static taint analysis for detecting sensor data leaks [55] could not detect these usages.

Web Sensor Usage. While this work focuses on user-installed third-party Android apps, we find that ARMOUR is also able to detect zero-permission sensor usage from the web, which has been shown to be another potential threat of sensor data leakage [20].

3.3 ARMOUR Implementation

Figure 3 summarizes the process of using ARMOUR to characterize sensor usage on an Android device. The core component is the ARMOUR app that continuously monitors in the background. The current implementation detects the usage of the three mostly used zero-permission sensors: accelerometers, gyroscopes, and magnetometers. Other sensors can be easily added to accommodate more specialized use cases.

Device Profiling. The profiling stage finds the minimum supported sampling rates f^i_{min}, which enables the data provider to set an appropriate threshold for the observed sampling rates f_{inst} to assert the use of each sensor in the detection stage. Each smartphone model with the same software OS version only needs to be profiled once. The ARMOUR app already implements a profiling mode to automatically achieve this. The profiling mode runs the app in the foreground, requests several sampling rates, and determines f^i_{min} by checking the actual received instant sampling rates. The whole profiling process takes less than 2 min.

Runtime Monitoring. Entering the monitoring mode, the ARMOUR app runs in the background while the data provider opens another app they want to analyze. The tested app can run either in the foreground or background. The ARMOUR app stores instant sampling rate data, i.e., the readings of the three sensors.

Instant Sampling Rate Processing & Detection. After collecting instant sampling rate data, ARMOUR calculates and examines a time series $f^i_{inst}(t)$ for each sensor i according to Equation 1. Our pilot testing observes that the fluctuation of instant sampling rates sometimes includes a few outliers, as shown in Figure 10. Such outliers are mostly caused by the transitions between different used sampling rates and may cause unstable sampling rates calculations. We thus implement a processing step to clean the time series. We define an outlier in f_{inst} as a sampling rate value with fewer than three consecutive occurrences and replace these values with its preceding or following value which is closer to the outlier value's magnitude. After cleaning the time series, we find that the time series data remains relatively stable with fluctuation errors always in the range of 0.4 Hz (Figure 9). The range of such fluctuations is affected by factors such as OEM and software variations. We thus empirically set the sensor usage detection threshold to $f^i_{min} + 0.5$. Following this methodology, the actual thresholds used for different devices can be adjusted after collecting sensor data in the device profiling phase. By default, this work declares sensor usage when $f^i_{inst}(t) > f^i_{thres} = f^i_{min} + 0.5$.

4 Evaluation

Our evaluation aims to explore the following research questions:

- *RQ1*: To what degree can ARMOUR reliably detect zero-permission sensor usage (Section 4.2)?
- *RQ2*: How does the overall landscape of real-world sensor usage look like (Section 4.3-4.5)?
- *RQ3*: What specific interesting sensor usage behaviors can ARMOUR reveal (Section 4.6)?

4.1 Experimental Setup

4.1.1 Dataset. We collected two Android application datasets, totaling 1,448 apps, to answer these questions. A Detection Performance evaluation dataset with 50 apps measures the precision and recall of ARMOUR in detecting sensor usage; A Sensor Usage dataset with 1,398 popular apps from Google Play allows us to analyze the widespread abuse of zero-permission sensors in real-world scenarios. By default, the evaluation uses the OnePlus Nord N200 phone with Android 12 as it has f^i_{min} of 1 Hz for gyroscopes and magnetometers and 5 Hz for accelerometers, providing a relatively large range of detectable instant sampling rates. Additionally, we also use the Samsung Galaxy S9 phone with Android 10 to analyze how different versions of Android, such as the introduction of the HIGH_SAMPLING_RATE_SENSORS permission in Android 12, could affect the pattern of sensor usage. The datasets thus contain APK/XAPK files for the two devices.

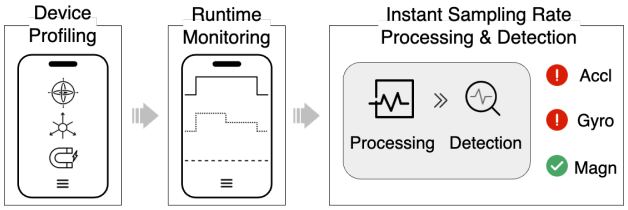

Figure 3: ARMOUR's workflow.

Detection Performance. Since there currently does not exist a ground-truth dataset for Android sensor usage, we constructed such a dataset consisting of 50 apps that are known to use sensors. We collected these 50 apps from Google Play by manually searching for apps that certainly use zero-permission sensors, such as those named (1) compass, magnetometers, EMF/metal detectors, etc., (2) acceleration meters, vibration meters, speedometers, etc., and (3) gyroscopes. These three categories of apps are known to use at least the phones' magnetometer, accelerometer, and gyroscope.

Sensor Usage. The dataset consists of 1398 most popular Google Play applications in the U.S. ranked by Appfigures [5]. APK/XAPK files are downloaded from APKPure and APKCombo. These apps span 35 different categories to ensure a broad representation of sensor usage patterns. The popularity and diversity of these applications aim to provide valuable insights into sensor access trends and potential abuse risks.

4.1.2 Procedure. The experiment tests one app in the dataset at a time, whose operations including installation, launching, etc., are automated by Python scripts running on a MacBook that utilizes the Android Debug Bridge (ADB) and the uiautomator2 tool [**?**]. The procedure of testing each app includes the following steps: (1) ARMOUR starts recording; it first runs for 5 seconds to ensure the recorded instant sampling rates equal f^i_{min} so as to confirm that no other software was accessing the sensors during the test. (2) The tested app starts running in the foreground for 15 seconds to examine its foreground sensor usage. (3) The tested app is then put in the background and runs for another 15 seconds. (4) The tested app is stopped and uninstalled and ARMOUR stops recording.

By default, the testing does not apply UI interactions to the tested apps. Since specific UI interactions and app events may trigger additional sensor usage, as will be shown in Section 4.6, this testing procedure provides a lower-bound baseline for the number of apps using sensors. Additionally, we captured periodic screenshots of

Table 1: Detected Sensor Usage of 1398 Most Popular Google Play Applications

Device	Android Version	Accelerometer	Gyroscope	Magnetometer	Any	All Three
Samsung Galaxy S9	10 (API 29)	639 (45.7%)	475 (34.0%)	359 (25.7%)	645 (46.1%)	310 (22.2%)
OnePlus Nord N200 5G	12 (API 31)	557 (39.8%)	485 (34.7%)	320 (22.9%)	583 (41.7%)	300 (21.5%)

the tested app using ADB to manually verify whether the app was successfully installed and launched.

4.2 Detection Performance & Overhead

This section evaluates ARMOUR's output when there is known sensor usage and the runtime overhead ARMOUR incurs.

4.2.1 Using Sensors. Our tests show that ARMOUR could successfully detect all sensor usage of the 50 apps. Among these apps, the minimum detected sampling rate of the used sensors is about 50 Hz, which corresponds to the constant values set by SENSOR_DELAY_GAME. The results provide evidence that ARMOUR can detect zero-permission sensor usage of most commercial apps with very high probabilities because apps are not likely to use f^i_{min} given the overly limited amount of information f^i_{min} can provide. A Frida-based dynamic analysis that aims to provide baseline measurements for commercial applications on a rooted phone will further confirm this observation (see Section 5).

4.2.2 Not Using Sensors. To further verify that ARMOUR does not cause false positives of sensor usage detection, we perform two additional tests. First, we ran ARMOUR in the background without any foreground apps to examine whether the Android OS activities could trigger sensor usage. Second, we ran a simple self-made application that does not use sensors to examine whether non-sensor app activities could trigger sensor usage. We ran both tests for 30 min continuously and 10 times each throughout the day, which provides a prolonged time frame that facilitates the assessment of long-term behaviors. The screen auto-rotation feature of the device was turned on in these tests, where sensors are used by the OS to detect change of phone orientations. ARMOUR did not mistakenly report any sensor usage in these cases. This confirms that changes in the instant sampling rates occur only when the SensorManager interface is explicitly invoked in user space.

4.2.3 Power & CPU Overhead. In the above tests of no foreground apps, we simultaneously profiled ARMOUR's CPU and memory usage using AccuBattery [1] and Android Studio's built-in Profiler. According to AccuBattery, running ARMOUR for background monitoring increased the battery usage from 2.0% of its total battery capacity per hour to 5.6% per hour, incurring a power consumption overhead that consumes approximately 2.6% of the battery per hour. This overhead is comparable to that induced by common background monitoring applications such as AccuBattery itself. The Profiler determined that CPU usage is light throughout ARMOUR's runtime out of the three categories of "Light", "Moderate", and "High". The relatively light power and CPU usage overheads are due to the fact that ARMOUR only needs to use a sampling rate of as low as 1 Hz to access sensors in the background. This makes it feasible to use ARMOUR not only for security research that does not face any limits of overheads but also for everyday background monitoring by average phone users who might care about overheads.

4.3 Popular Google Play Applications

Table 4 summarizes the percentage of apps using each sensor, showing zero-permission sensor usage is very common both before and after Android's change in zero-permission sensor regulation in Android 12. We additionally analyzed 150 apps manually that were randomly selected from the dataset and found that *none of these apps explicitly stated their purposes for using zero-permission sensors*, either in the app UI or Google Play webpage introduction.

On the Samsung Galaxy S9 device with Android Android 10, 46.1% of the 1398 apps use at least one of the three sensors. The accelerometer is used the most, followed by the gyroscope, with the magnetometer being the least used (25.7%). The pattern of sensor usage on the OnePlus Nord N200 device with Android 12 remains consistent with slight decreases in accelerometer and magnetometer usage and a slight increase in gyroscope usage. Given that the Samsung Galaxy S9 and OnePlus Nord N200 devices have different f^{accel}_{min} which result in 5.5 Hz and 1.5 Hz accelerometer usage detection thresholds respectively, we also tested setting f^{accel}_{thrs} of the Samsung device to 5.5 Hz to control the factor. This produces an accelerometer usage percentage of 42.3% which is 3.4% lower than using a 1.5 Hz threshold, showing that a small portion of apps used low accelerometer sampling rates in the range 1.5 to 5.5 Hz. It is also an observed common pattern that the accelerometer, gyroscope, and magnetometer are often accessed together. More than 20% of the 1398 apps, i.e., over 50% of all apps using sensors, are found to be using all three sensors simultaneously.

4.3.1 App Categories. Figure 4 further provides a breakdown of these usages by app categories. Table 4 provides more detailed data with several representative apps in each category. While most categories of apps follow the pattern that the accelerometer and magnetometer are the most and least used sensors among the three, there are some exceptions. For example, the categories Art/Design and Personalization use gyroscope the most. In addition, Shopping and Travel/Local use magnetometers more than gyroscopes and accelerometers, which is likely due to localization services these apps provide that depend on magnetometer data for navigation.

The most outstanding problem observed is the unclear and unexplained purpose of sensor usage in many app categories. Surprisingly, categories that people often do not expect to have apparent dependencies on sensor-collected information, such as apps in Finance, Dating, Book/Reference, Shopping, Weather, etc., actually heavily depend on accelerometer and gyroscope data for unspecified purposes. For example, more than half of the Book/Reference apps use accelerometer and gyroscope data but identifying the rationale behind this usage is challenging. Furthermore, although the results of certain categories such as Health/Fitness seem to align with expected sensor usage, a closer look at the individual apps raises questions, such as why the subcategory of exercise planning apps that are not supposed to monitor user motions still frequently collect sensor data.

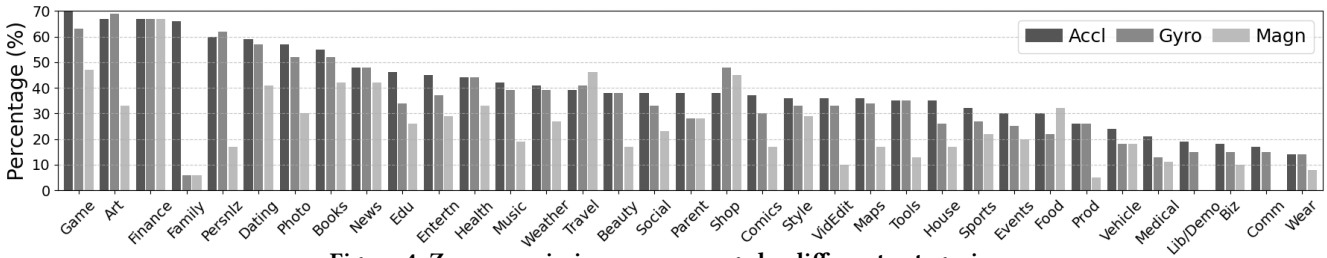

Figure 4: Zero-permission sensor usage by different categories.

In addition, results such as the sensor usage by 67% of the Finance apps collecting data from all three sensors reveals that real-world applications might already be using sensor data for identification. Our literature review shows that Finance apps use sensor data information for authenticating and tracking users and devices [6, 56]. While the collected sensor data is theoretically used for benign purposes of enhancing security in these cases, it also echoes the findings of prior research that real-world applications have the capability of exploiting sensor data for sensitive information [29].

4.4 Sampling Rate Distribution

This section analyzes the sensor sampling rates used by popular Google Play apps and the impact of high sampling rate permission introduced by Android 12.

Figure 6 shows the histogram of the maximum detected sampling rates of each app for the three sensors in the two Android versions respectively. We observe that while many of the app developers tend to request sampling rates defined by the Android-defined constants `SENSOR_DELAY_FASTEST/GAME/UI/NORMAL`, which correspond to sampling rates around 400, 50, 15, and 5 Hz, there are many apps manually requesting other sampling rates such as 200 and 100 Hz.

The highest sampling rate defined by `SENSOR_DELAY_FASTEST` corresponds to 416 Hz for the accelerometer and the gyroscope and 100 Hz for the magnetometer of the two phones. In Android 10, 20.0%, 25.5%, and 34.5% apps used these highest sampling rates for the three sensors respectively, with 120 out of 130 (92.3%) applications simultaneously accessing the highest frequency for all three sensors. These numbers dropped to 0.8%, 0.4%, and 32.0% in Android 12. Out of the 128 apps that used `SENSOR_DELAY_FASTEST` for accelerometer and gyroscope in Android 10, only 5 of them took the effort of declaring the high sampling rate permission in Android 12 to keep using it. On the contrary, the `SENSOR_DELAY_FASTEST` usage for the magnetometer only observed a negligible decrease because 100 Hz is not regulated by the `HIGH_SAMPLING_RATE_SENSORS` permission. The results indicate that the regulation introduced by Android 12 effectively limited certain unnecessary high sampling rate usage.

We found that `SENSOR_DELAY_GAME` responds to a sampling rate of 52 Hz for accelerometers and gyroscopes in Android 12. Interestingly, however, the highest sampling rate apps can get without declaring the `HIGH_SAMPLING_RATE_SENSORS` permission in Android 12 is 206 Hz on the OnePlus device and about 21.0% of apps used this sampling rate. Requesting sampling rates higher than 206 Hz without declaring the `HIGH_SAMPLING_RATE_SENSORS` permission in Android 12 and above results in compiling errors. This phenomenon suggests two insights. First, many apps tend to use the highest available sampling rates without declaring the `HIGH_SAMPLING_RATE_SENSORS` permission even if they have to manually test and specify feasible sampling rate values. Second, while Android 12 and above officially specify that the maximum available sampling rates without declaring `HIGH_SAMPLING_RATE_SENSORS` should be no more than 200 Hz [27], the implementation on the OnePlus device of 206 Hz sampling rates suggests *the potential for manufacturers to violate or bypass this requirement. It remains unclear how Android could enforce this requirement to regulate manufacturers.*

4.5 Foreground & Background Sensor Usage

Our results show that apps tend to have distinct foreground and background sensor usage patterns, especially across different app categories. Out of the 583 apps that used sensors, all of them had foreground access while 168 of them (28.8%) had background access.

4.5.1 Foreground Usage. Although, at first glance, the high percentage of foreground sensor usage might be thought of as being associated with user interactions, our experiments already eliminated the impact of this factor by not having any UI inputs in the testing procedure as mentioned in Section 4.1.2. Thus, the sensor data collections occured without any user interaction after opening the application. For example, Figure 7 shows three typical types of foreground activities after the apps start that are not supposed to have reasonable sensor-related operations, such as asking users to agree to the apps' policies, showing a welcome message, and prompting users to log in. Notably, we found that about 65% of the apps with foreground sensor access paused at these pages.

4.5.2 Background Usage. The highest background sensor usage ratios were observed in Productivity (60.0%), Communication (57.1%), and Games (51.4%). While Games apps such as racing are expected to heavily depend on sensor data for user hand gesture tracking, the majority of their background usage appears unexplained as any background activities do not change the states of the apps. Such background usage is thus likely to be negligent sensor abuse due to poor mobile device resource management practices. The high background usage ratios in Productivity and Communication appear less intuitive. On the contrary, the Android-Wear category that provides services to resource-constrained wearable devices demonstrated no (0%) background sensor usage. This suggests that these applications, at a minimum, require explicit user interaction to activate sensors rather than maintaining passive access in the background, making them a good example of responsible sensor usage in background stages.

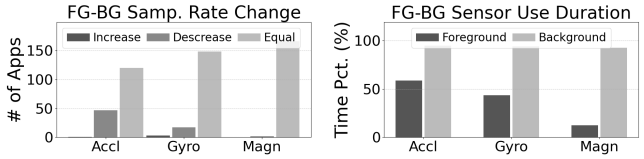

Figure 5: Background sensor usage tends to have decreasing or equal sampling rates (left) but higher durations (right).

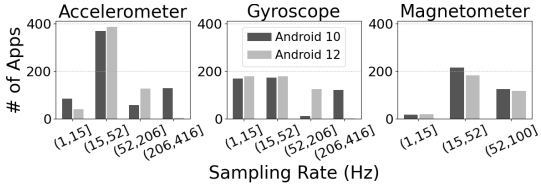

Figure 6: Distribution of Google Play apps' zero-permission sensor sampling rates before and after Android 12.

4.5.3 Foreground-Background Transition. Comparing sensor usage in the foreground and background shows that apps generally use lower sensor sampling rates in the background, while simultaneously extending the duration of sensor access, as shown in Figure 5. The three types of zero-permission sensors exhibit significantly higher peak access sampling rates in the foreground, with a wide interquartile range and numerous high sampling rate outliers. Once transitioned to the background, the maximum frequency drops substantially. This verifies apps' common behaviors of shifting toward persistent, long-term monitoring in the background without user awareness.

4.6 Case Study

This section further discusses several use cases of ARMOUR that revealed interesting sensor usage behaviors during our examination of individual apps.

Persistent Sensor Access After App Termination. It was surprising to observe that certain apps could continue to access zero-permission sensors after being killed in GUI or even force-stopped using ADB. More than 50 (3.6%) apps exhibited reproducible persistent sensor activity after termination on the OnePlus phone. However, running the same apps on a Pixel 3 smartphone did not show the same persistent sensor usage. We believe this OEM variation is caused by the fact that Android allows manufacturers to customize its behaviors of process life-cycle and background service handling [28]. Additionally, other factors such as specific sensor hardware and drivers could also cause different sensor access termination behaviors. This indicates further challenges of enforcing strict sensor access policies across different platforms.

Extended Running and User-triggered Sensor Usage. The evaluations so far ran each app for only 15 seconds in the foreground without any user interactions. This setting provides a lower-bound measurement, indicating that more extensive sensor data collection and abuse could exist. To confirm this, we manually tested 35 randomly selected apps, one from each category. 18 of them had foreground sensor usage detected in previous tests. We had normal GUI interactions with each of these apps for 5 minutes continuously. The tests found additional sensor usage triggered by user interactions in 14 out of these 18 apps. The most common trigger is user logins (6 apps), such as hitting the "Continue With Google" button. This suggests the sensor data is used as an information source for authentication. Other triggers also include accepting policies, granting permissions, and other app-specific actions. Furthermore, user interactions triggered sensor usage in 5 of the 17 apps that did not use sensors in the 15-second foreground tests. This suggests future research could integrate ARMOUR with automated GUI testing frameworks for large-scale measreument.

Furthermore, ARMOUR's capability of monitoring time-variant sensor usage correlated with user interactions further enables analysis of sensor-based exploitation including *shake-triggered advertisement* found on apps such as Bilibili–one of the most popular video sharing platforms. After a brief opening page, the app enters a shake-ad activity that continuously collects sensor data for 5 seconds. The shake-ad page has a concealed prompt of "shake the phone to enter Taobao". Even when users did not notice this prompt, users' minor involuntary hand movements could be detected by the accelerometer and gyroscope with sampling rates of 15 Hz (SENSOR_DELAY_UI) and be regarded as users actively shaking their phones, thus redirecting to the Taobao shopping app by Alibaba (Figure 8). The Taobao app continues to collect data from the two sensors for unknown purposes, at 52 Hz (SENSOR_DELAY_GAME). This discovered pattern confirms what has been hypothesized in previous research such as triggering a malicious app via sensors [50].

Third-party Libraries. The observed common sensor access patterns in different applications suggest a hypothesis that some zero-permission sensor usage could be caused by shared third-party libraries. To provide preliminary insights, we decompiled representative apps using jadx [52] and analyzed their obfuscated source codes. We found a large portion of apps with the unique pattern of accessing sensors for a very short time at app startup, with the highest possible sampling rates. Reading their decompiled code could accurately identify this sensor usage by Appsflyer [6], a marketing analytics service that helps app developers track and optimize user acquisition campaigns. Appsflyer records data from the three sensors for 500 ms, and produce a hash from the saved data to match users/devices. Our dataset shows that about 49% of the apps using sensors have included the Appsflyer library. In contrast, PayPal Magnes [56], which is a widely used mobile payment SDK for collecting real-time device data to help detect fraud, continuously saves collected sensor data at low sampling rates (about 20 Hz) to JSON files during its operation, and then sends the files to its remote servers. About 6% of the apps using sensors included code of PayPal Magnes service. Additionally, the sensor access patterns of multiple libraries may be superimposed. These results also suggest possible follow-up research of identifying third-party libraries by analyzing sensor behaviors detected by ARMOUR from user space.

5 Discussion

This section discusses the limitations and possible future research directions revealed by ARMOUR.

Sensor Usage Ground Truth & Baseline. As Section 4 explained, the lack of a ground truth presents a major roadblock for

Figure 7: Examples of unexpected zero-permission sensor usage when apps start in the foreground.

zero-permission sensor monitoring. The datasets collected by our work take the first step toward bridging this gap, but unavoidably faces limitations of lacking a baseline that ARMOUR could compare with. In particular, certain apps could still be using the minimum supported sampling rates f^i_{min} and thus bypass the detection of ARMOUR. Aiming to probe how to provide such a baseline, our work explored applying the Frida dynamic instrumentation toolkit [46] on a rooted smartphone to monitor the invocation of the onSensorChanged callbacks. The results show that out of the 830 apps for which ARMOUR did not detect zero-permission sensor usage, only 38 (4.49%), 3 (0.35%), and 10 (1.18%) of them use the accelerometer, gyroscope, and magnetometer at their respective f^i_{min}. This confirms that commercial applications mostly use non-minimum sampling rates to collect more sensor data. The higher percentage of accelerometer is due to a relatively high $f^{accl}_{min} = 5$. To further reduce the risks of undetected sensor usage at the minimum sampling rates, we recommend that privacy-aware manufacturers implement $f^i_{min} = 1$ for all sensors on future devices. It is also worth noting that Frida only provides a baseline for comparison instead of the ultimate ground truth due to the possible Runtime Android Self Protection measures (RASP) that may prevent dynamic instrumentation or running apps on rooted devices. This motivates future works of comprehensive ground truth development.

Sensor Usage Attribution. Our results so far reveal important follow-up questions, such as how to distinguish between benign and malicious sensor access patterns. Determining whether a zero-permission sensor is exploited to maliciously acquire private information that users do not want to share poses significant new challenges compared to prior research on identifying malicious Android applications. This is because, unlike IMEI or other textual data targeted by conventional malicious apps that can be directly associated with entities of private information after being detected in the saved files or network packets, the captured raw sensor data needs to be processed by the sensor-based malicious apps themselves to extract semantic signals that correlate with private information (Section 6.1). Thus, identifying a sensor-based malicious app is equivalent to completely reverse engineering the app's sensor-related functionalities. This challenge could be further amplified when the apps use a server-side exploitation model, where only raw sensor packets are transmitted to a remote server for black-box processing. These challenges motivate dedicated future research that integrates reverse engineering, automated large-scale static analysis, root-based dynamic analysis, and user studies. In the extremely challenging case of server-side black-box processing, we believe that advanced methods such as differential behavior analysis [18] need to be developed to empirically probe whether collected sensor data is used for malicious or benign purposes.

Figure 8: ARMOUR detects the sensor usage behind emerging shake-ad features of some apps.

Although these topics fall out of the scope of this work, the sensor usage patterns measured by ARMOUR could serve as motivating examples and baselines for future research.

Environments & Factors. Our evaluations demonstrated the effectiveness of ARMOUR in lab environments with a relatively small number of varying factors. Our measurement analysis focused on Android 12, whose sensor-related policies have remained consistent up to Android 15 at the time of writing. Without further policy changes, running ARMOUR on other devices will not be affected by the newer Android versions. Instead, we found that specific OEM configurations, such as different minimum supported sampling rates (e.g., Table 3), could have larger impact on ARMOUR's results. We believe this encourages future research by dedicated security analysts that utilizes ARMOUR to perform in-depth app or device-wise analysis. Potentially, an online community and database could be created to crowdsource ARMOUR's measurements on various devices.

Future Sensor Privacy Enhancement. While the current ARMOUR app can be readily used by lay users for sensor usage notification (as shown in Figure 1), additions to the app's UI functionality will further improve its usability as a privacy enhancement tool. A limitation of the current ARMOUR design is the requirement of running a single app besides the monitor. In daily monitoring scenarios, detection performance could degrade if multiple third-party apps run concurrently because ARMOUR can only ascertain that at least one app is using sensors. It is possible to address this with more sophisticated ARMOUR designs, which could require additional permissions, such as UsageStats, to monitor when each app is active in the foreground/background to create a correlation engine that maps temporal patterns of sensor usage with app activities. As a user-trusted app, ARMOUR may utilize Android's accessibility feature [26] to monitor GUI activities that trigger sensor usage and thus pre-build a signature database for different apps' sensor access patterns.

Our findings also motivate the need for enhanced privacy protections after detecting zero-permission sensor abuses. On the policy level, we suggest platform providers such as Google Play implement knowledge-based auditing and stricter review processes to regulate unnecessary sensor access. This could require adding explanations of sensor data usage, as a standard procedure for generating SBOM [16] for Android and similar ecosystems. We have disclosed the identified sensor abuse problems to Google and aim to provide detailed mitigation suggestions in further interactions. From a technical perspective, future research could investigate how

to filter out sensitive information or selectively inject controlled noise into sensor data streams to preserve utility while mitigating privacy leakage [13]. Although some related works (Section 6.2) explored similar approaches, finding more deployable and scalable mitigation techniques that may not require OS modifications or root privileges remains an open challenge. More fine-grained permission models for sensor data are needed, though helping users understand privacy implications without adding cognitive burden remains challenging [10]. This is further complicated by sensor usage attribution difficulties identified in this work. We see significant opportunities for research into effective and usable privacy-preserving sensing mechanisms.

6 Related Work

This section introduces prior research investigating the threats of zero-permission sensor usage and the possible ways of mitigation.

6.1 Zero-permission Sensor Exploitation

Research has extensively shown threats posed by zero-permission sensors. [51] provides a survey that summarizes these existing problems in a systematic way. In 2014, Gyrophone [39] first analyzed how sound generated by electronic speakers in the vicinity of smartphones can be captured by phone applications accessing the phones' gyroscope readings. A series of follow-up research explored the possibility of using accelerometers for speech eavesdropping [4, 8, 31]. Similarly, users inputting texts or PIN on the phone touchscreen or walking with the phone cause unique and discernible phone motions that can be captured by zero-permission motion sensors [15, 32, 34, 42]. Magnetometers are another type of the most analyzed zero-permission sensors for leaking information. Several recent works discovered that smartphones' electromagnetic emissions can be received by the built-in magnetic sensors, which contain signals that enable applications to infer specific phone CPU and display activities [17], or device locations [12]. While these prior works verified the privacy impact of malicious zero-permission sensor exploitation, it remains unknown to what degree commercial applications have been using data from these sensors and whether the measured usage patterns could reveal potential abusive problems [50]. This work seeks to address this gap by developing ARMOUR that measure the first collected dataset of zero-permission sensor usage detection.

6.2 Android Sensor Analysis & Protections

Analyzing and protecting against the possible exploitation of Android sensors is an emerging research field motivated by the observed zero-permission sensor problems.

Static Analysis-based Usage Detection. Although there has been a large body of taint-analysis such as FlowDroid [7], these early works did not consider sensors as an input source that could leak information. Liu et al. [33] developed a tool SDFDroid, which disassembles APK files to smali code files using Apktool and looks for sensor listener registration and data receiving callbacks. SDFDroid has the limitations of not performing well on applications with code obfuscation and native code, and cannot detect which exact sensors are used. A later work by Sun et al. [55] extends FlowDroid to detect leaks with sensor data sources. Despite its efforts to develop a rule-based approach to identify the sensor types, its approach is still limited by obfuscations and native codes, and only works in the case of a single sensor type being registered. These gaps show the common limitations of static analysis-based methods for detecting Android sensor usage. Furthermore, these works' aims also differ from this research. While they focus on developer-side analysis which requires sophisticated steps of building development environments and code dissembling, our work aims to provide a ready-to-use tool that can be utilized by any user and researcher.

OS and App Instrumentation. Some other works seek to provide better security against zero-permission sensors by directly modifying the existing Android OS or third-party applications. 6thSense [50] developed a context-aware sensor-based attack detection framework that monitors sensor data, infers the current use context, and then decides whether the current sensor use might be malicious. To detect the use context, 6thSense needs to acquire all sensor readings sent to each application, which requires 6thSense to be built into the operating system. Sensor Guardian [9] and SemaDroid [58], on the other hand, choose to directly control applications' access to sensors by inserting hooks into applications' Dalvik byte-codes or modifying OS implementations to enforce additional control policies at runtime. This requires developers to build a sophisticated development chain in order to modify and control Apps and is limited to use by specialized experts. Furthermore, this approach does not work with apps implementing RASP techniques. In contrast to these approaches, ARMOUR focuses on the user space sensor usage detection without modifications to the existing Android OS or requiring root privilege. This provides complementary protection capabilities when root and expert privileges are not available.

7 Conclusion

This work investigated monitoring zero-permission sensor usage on Android and developed ARMOUR, a user-space tool allowing researchers and users to analyze when, which, and at what sampling rates third-party apps access these sensors. Our evaluation with 1,448 commercial apps revealed widespread, unjustified sensor usage across multiple app categories, highlighting risks from unregulated access and limitations in current Android security policies. Our findings call for improved regulations and advanced privacy tools to address growing threats. We present ARMOUR and our dataset as an important step toward mitigating the information asymmetry between data collectors and providers.

Acknowledgments

We appreciate the insights and remarks from our reviewers. This research was supported in part by the National Science Foundation (NSF) Industry-University Cooperative Research Centers Program, CHEST, under grant IUCRC-1916762, and by the NSF award 2330264. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily of the NSF.

References

[1] 2023. AccuBattery. https://accubatteryapp.com/
[2] 2025. Access Dots - iOS cam/mic/gps. https://play.google.com/store/apps/details?id=you.in.spark.access.dots&hl=en_US

[3] 2025. Safe Dot. https://github.com/kamaravichow/safe-dot-android?tab=readme-ov-file

[4] S Abhishek Anand, Chen Wang, Jian Liu, Nitesh Saxena, and Yingying Chen. 2019. Spearphone: A speech privacy exploit via accelerometer-sensed reverberations from smartphone loudspeakers. *arXiv preprint arXiv:1907.05972* (2019).

[5] Appfigures. 2025. Top Ranked Google Play Apps. https://appfigures.com/top-apps/google-play/united-states/top-overall

[6] Appsflyer. 2025. Pay for real customers, not bots, with advanced sensor analysis. https://www.appsflyer.com/products/fraud-protection/

[7] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.

[8] Zhongjie Ba, Tianhang Zheng, Xinyu Zhang, Zhan Qin, Baochun Li, Xue Liu, and Kui Ren. 2020. Learning-based Practical Smartphone Eavesdropping with Built-in Accelerometer.. In *NDSS*, Vol. 2020. 1–18.

[9] Xiaolong Bai, Jie Yin, and Yu-Ping Wang. 2017. Sensor Guardian: prevent privacy inference on Android sensors. *EURASIP Journal on Information Security* 2017 (2017), 1–17.

[10] Michael Benisch, Patrick Gage Kelley, Norman Sadeh, and Lorrie Faith Cranor. 2011. Capturing location-privacy preferences: quantifying accuracy and user-burden tradeoffs. *Personal and Ubiquitous Computing* 15 (2011), 679–694.

[11] David Berend, Shivam Bhasin, and Bernhard Jungk. 2018. There goes your pin: Exploiting smartphone sensor fusion under single and cross user setting. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. 1–10.

[12] Kenneth Block and Guevara Noubir. 2018. My magnetometer is telling you where i've been? a mobile device permissionless location attack. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 260–270.

[13] Connor Bolton, Kevin Fu, Josiah Hester, and Jun Han. 2020. How to curtail oversensing in the home. *Commun. ACM* 63, 6 (2020), 20–24.

[14] Connor Bolton, Yan Long, Jun Han, Josiah Hester, and Kevin Fu. 2023. Characterizing and Mitigating Touchtone Eavesdropping in Smartphone Motion Sensors. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 164–178.

[15] Liang Cai and Hao Chen. 2011. TouchLogger: inferring keystrokes on touch screen from smartphone motion. *HotSec* 11, 2011 (2011), 9.

[16] L Jean Camp and Vafa Andalibi. 2021. Sbom vulnerability assessment & corresponding requirements. *NTIA Response to Notice and Request for Comments on Software Bill of Materials Elements and Considerations* (2021).

[17] Yushi Cheng, Xiaoyu Ji, Wenyuan Xu, Hao Pan, Zhuangdi Zhu, Chuang-Wen You, Yi-Chao Chen, and Lili Qiu. 2019. Magattack: Guessing application launching and operation via smartphone. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 283–294.

[18] Andrea Continella, Yanick Fratantonio, Martina Lindorfer, Alessandro Puccetti, Ali Zand, Christopher Kruegel, Giovanni Vigna, et al. 2017. Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis.. In *NDSS*, Vol. 17. 10–14722.

[19] International Data Corporation. 2023. Smartphone Market Share. https://www.idc.com/promo/smartphone-market-share

[20] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1515–1532.

[21] Erhan Davarci, Betul Soysal, Imran Erguler, Sabri Orhun Aydin, Onur Dincer, and Emin Anarim. 2017. Age group detection using smartphone motion sensors. In *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, 2201–2205.

[22] Android Developers. 2018. Android 9 behavior changes. https://developer.android.com/about/versions/pie/android-9.0-changes-all#bg-sensor-access

[23] Android Developers. 2021. Android 12 behavior changes. https://developer.android.com/about/versions/12/behavior-changes-12#motion-sensor-rate-limiting

[24] Android Developers. 2023. Manifest.permission: HIGH_SAMPLING_RATE_SENSORS. https://developer.android.com/reference/android/Manifest.permission#HIGH_SAMPLING_RATE_SENSORS

[25] Android Developers. 2023. Monitoring Sensor Events. https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-monitor

[26] Android Developers. 2025. Create your own accessibility service. https://developer.android.com/guide/topics/ui/accessibility/service

[27] Android Developers. 2025. Sensor Rate-Limiting. https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview#sensors-rate-limiting

[28] Android Developers. 2025. System restrictions on background work. https://developer.android.com/develop/background-work/background-tasks/bg-work-restrictions?utm_source=chatgpt.com#user-initiated-restrictions

[29] e Foundation. 2024. PayPal: Data transfer to over 600 third-party companies + metadata. https://community.e.foundation/t/paypal-data-transfer-to-over-600-third-party-companies-metadata/61888

[30] Gabriella M Harari, Sandrine R Müller, Min SH Aung, and Peter J Rentfrow. 2017. Smartphone sensing methods for studying behavior in everyday life. *Current opinion in behavioral sciences* 18 (2017), 83–90.

[31] Pengfei Hu, Hui Zhuang, Panneer Selvam Santhalingam, Riccardo Spolaor, Parth Pathak, Guoming Zhang, and Xiuzhen Cheng. 2022. Accear: Accelerometer acoustic eavesdropping with unconstrained vocabulary. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1757–1773.

[32] Wei-Han Lee and Ruby B Lee. 2017. Implicit smartphone user authentication with sensors and contextual machine learning. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 297–308.

[33] Xing Liu, Jiqiang Liu, Wei Wang, Yongzhong He, and Xiangliang Zhang. 2018. Discovering and understanding android sensor usage behaviors with data flow analysis. *World Wide Web* 21 (2018), 105–126.

[34] Yan Long and Kevin Fu. 2022. Side Auth: Synthesizing Virtual Sensors for Authentication. In *Proceedings of the 2022 New Security Paradigms Workshop*. 35–44.

[35] Yan Long, Qinhong Jiang, Chen Yan, Tobias Alam, Xiaoyu Ji, Wenyuan Xu, and Kevin Fu. 2024. EM Eye: Characterizing electromagnetic side-channel eavesdropping on embedded cameras. (2024).

[36] Yan Long, Pirouz Naghavi, Blas Kojusner, Kevin Butler, Sara Rampazzi, and Kevin Fu. 2023. Side eye: Characterizing the limits of pov acoustic eavesdropping from smartphone cameras with rolling shutters and movable lenses. In *2023 IEEE symposium on security and privacy (SP)*. IEEE, 1857–1874.

[37] Anindya Maiti, Ryan Heard, Mohd Sabra, and Murtuza Jadliwala. 2018. Towards inferring mechanical lock combinations using wrist-wearables as a side-channel. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 111–122.

[38] Sumit Majumder and M Jamal Deen. 2019. Smartphone sensors for health monitoring and diagnosis. *Sensors* 19, 9 (2019), 2164.

[39] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. 2014. Gyrophone: Recognizing speech from gyroscope signals. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 1053–1067.

[40] Reham Mohamed, Habiba Farrukh, Yidong Lu, He Wang, and Z Berkay Celik. 2023. iStelan: Disclosing Sensitive User Information by Mobile Magnetometer from Finger Touches. *Proceedings on Privacy Enhancing Technologies* 2 (2023), 79–96.

[41] Sashank Narain, Triet D Vo-Huu, Kenneth Block, and Guevara Noubir. 2016. Inferring user routes and locations using zero-permission mobile sensors. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 397–413.

[42] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. 2012. Accessory: password inference using accelerometers on smartphones. In *proceedings of the twelfth workshop on mobile computing systems & applications*. 1–6.

[43] Hao Pan, Lanqing Yang, Honglu Li, Chuang-Wen You, Xiaoyu Ji, Yi-Chao Chen, Zhenxian Hu, and Guangtao Xue. 2021. Magthief: Stealing private app usage data on mobile devices via built-in magnetometer. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 1–9.

[44] Jan Pennekamp, Martin Henze, and Klaus Wehrle. 2017. A survey on the evolution of privacy enforcement on smartphones and the road ahead. *Pervasive and Mobile Computing* 42 (2017), 58–76.

[45] Android Open Source Project. 2022. Android Sensor Stack. https://source.android.com/docs/core/interaction/sensors/sensor-stack#framework

[46] Ole André V. Ravnås. 2025. Frida: A world-class dynamic instrumentation toolkit. https://frida.re/docs/android/

[47] Google Samples. 2021. High sensor sampling rate code warning. https://googlesamples.github.io/android-custom-lint-rules/checks/HighSamplingRate.md.html

[48] Nina Shamsi, Yan Long, and Kevin Fu. 2023. EyeHearYou: Probing Location Identification via Occluded Smartphone Cameras and Ultrasound. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 36–47.

[49] Cong Shi, Xiangyu Xu, Tianfang Zhang, Payton Walker, Yi Wu, Jian Liu, Nitesh Saxena, Yingying Chen, and Jiadi Yu. 2021. Face-Mic: inferring live speech and speaker identity via subtle facial dynamics captured by AR/VR motion sensors. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 478–490.

[50] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2017. 6thSense: A Context-aware Sensor-based Attack Detector for Smart Devices.. In *USENIX Security Symposium*. 397–414.

[51] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. 2021. A survey on sensor-based threats and attacks to smart devices and applications. *IEEE Communications Surveys & Tutorials* 23, 2 (2021), 1125–1159.

[52] GitHub skylot. 2025. jadx: Dex to Java decompiler. https://github.com/skylot/jadx

[53] Raphael Spreitzer. 2014. Pin skimming: exploiting the ambient-light sensor in mobile devices. In *Proceedings of the 4th ACM Workshop on Security and Privacy*

in Smartphones & Mobile Devices. 51–62.
[54] Ke Sun, Chunyu Xia, Songlin Xu, and Xinyu Zhang. 2023. StealthyIMU: Stealing permission-protected private information from smartphone voice assistant using zero-permission sensors. Network and Distributed System Security (NDSS) Symposium.
[55] Xiaoyu Sun, Xiao Chen, Kui Liu, Sheng Wen, Li Li, and John Grundy. 2021. Characterizing sensor leaks in android apps. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 498–509.
[56] Verifone. 2024. PayPal Magnes. https://verifone.cloud/docs/online-payments/apm/paypal-ecom/paypal-magnes
[57] Yi Wu, Cong Shi, Tianfang Zhang, Payton Walker, Jian Liu, Nitesh Saxena, and Yingying Chen. 2023. Privacy leakage via unrestricted motion-position sensors in the age of virtual reality: A study of snooping typed input on virtual keyboards. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3382–3398.
[58] Zhi Xu and Sencun Zhu. 2015. Semadroid: A privacy-aware sensor management framework for smartphones. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. 61–72.
[59] Yuqing Yang, Mohamed Elsabagh, Chaoshun Zuo, Ryan Johnson, Angelos Stavrou, and Zhiqiang Lin. 2022. Detecting and Measuring Misconfigured Manifests in Android Apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3063–3077.

A Appendix: Supplementary Materials

Table 2: Examples of Possible Information Leakage

Category	Sensor	Info. Leakage Example
Motion	Gyroscope	Speech audio [39, 54]
	Gyroscope	Lock information [37]
	Accelerometer	Speech reconstruction [8, 31]
	Accelerometer	Touchscreen input [11, 42]
	Accl. + Gyro.	User age & identity [21, 32]
	Accl. + Gyro.	Keystroke [15, 57]
Position	Magnetometer	User location [12, 41]
	Magnetometer	App activity [17, 40]

Table 3: Examples of Android Phones' Parameters

Phone Model	Android Version	f_{min}^{accl}	f_{min}^{gyro}	f_{min}^{magn}
Google Pixel 3	12	5	1	1
Google Pixel 5	12	5	1	1
Google Pixel 6	13	7	2	1
OnePlus Nord N200	12	5	1	1
Samsung Galaxy S9	10	1	1	1
Samsung Galaxy S20	13	1	1	1

*The unit of minimum supported sampling rates is Hz.

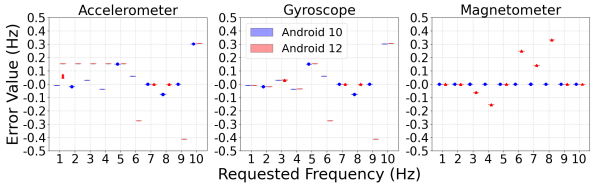

Figure 9: Variation ranges of received instant sampling rates when apps request different sampling rates. The sampling rates remain stable with errors smaller than 0.4 Hz.

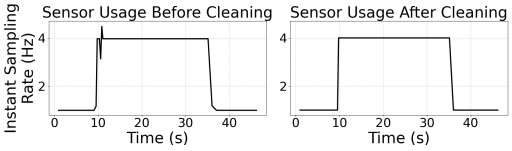

Figure 10: Example of the effect of outlier cleaning in f_{inst}.

Table 4: Zero-permission Sensor Usage Pattern of Different Categories of Popular Google Play Apps

Category (# apps)	Accl.	Gyro.	Magn.	All	None	Examples of Apps Accessing Sensors
Games (38)	92%	63%	47%	47%	8%	Roblox; Bus Out; Hole.io
Art/Design (39)	67%	69%	33%	33%	28%	Arvin® – AI Logo Maker; Creati AI Photo Generator; Sketchbook Lite
Finance (6)	67%	67%	67%	67%	33%	Venmo; testerup – earn money; PayPal – Pay, Send, Save
Family (35)	66%	6%	6%	6%	34%	Bluey: Let's Play!; Floof – My Pet House; Disney Coloring World
Personalization (47)	60%	62%	17%	17%	36%	Launcher OS™; Easy Homescreen; Neon Love Theme
Dating (37)	59%	57%	41%	41%	41%	Dating & Chat Online; Dating & Chat – iHappy; Dating & chat – Likerro
Photography (44)	57%	52%	30%	30%	43%	Meitu; Skylight; Meta View
Books/Reference (40)	55%	52%	42%	42%	45%	WebNovel; NovelFlow; Holy Bible Light
News/Magazines (48)	48%	48%	42%	40%	50%	Newsmax; Quick News; News Today
Education (35)	46%	34%	26%	23%	51%	ClassDojo; Dino Coloring Game; Imprint: Learn Visually
Entertainment (38)	45%	37%	29%	29%	55%	Xbox; STARZ; Reel Rush
Health/Fitness (39)	44%	44%	33%	33%	54%	JustFit-Lazy Workout; Pilates Workout; Pedometer-Step Counter
Music/Audio (36)	42%	39%	19%	19%	56%	Radio FM; Ringtones for Android™; Pocket FM: Audio Series
Weather (44)	41%	39%	27%	27%	57%	Weather&Radar; Know Weather: Live Radar; Weather Forecasts&Radar
Travel/Local (41)	39%	41%	46%	39%	54%	earnify; Fly Delta; Disneyland®
Beauty (47)	38%	38%	17%	15%	60%	Barber Chop; GlossGenius; Picture Editor
Social (39)	38%	33%	23%	23%	62%	Facebook; Instagram; Letterboxd
Parenting (47)	38%	28%	28%	28%	62%	Alli360 by Kids360; Pregnancy Tracker: amma; Bark – Parental Controls
Shopping (29)	38%	48%	45%	31%	48%	Kroger; Lowe's; Circle K
Comics (46)	37%	30%	17%	17%	63%	K MANGA; Pocket Toons; Key Collector Comics
Lifestyle (42)	36%	33%	29%	26%	62%	AARP Now; Pinterest; Gold Town
Video-Players (42)	36%	33%	10%	10%	64%	Rumble; MX Player; Video Maker
Maps/Navigation (47)	36%	34%	17%	17%	62%	Phone Tracker; Roadie Driver; Bolt: Request a Ride
Tools (46)	35%	35%	13%	13%	65%	Manic; Wodfix Max; Neat Manager – AntiVirus
House/Home (46)	35%	26%	17%	13%	65%	Merkury Smart; Apartment List; PadSplit: Rooms for rent
Sports (37)	32%	27%	22%	22%	68%	NFL; GameChanger; NFL Network
Events (44)	30%	25%	20%	18%	68%	Timeleft; Posh – Social Experiences; Bridebook – Wedding Planner
Food/Drink (40)	30%	22%	32%	18%	55%	Wawa; Crumbl; Subway®
Productivity (39)	26%	26%	5%	5%	74%	AI Chatbot – Nova; PDF Reader – PDF Viewer; Email Lite – Smart Mail
Auto/Vehicles (38)	24%	18%	18%	18%	76%	PayByPhone; Fuel Forward; CARFAX Car Care App
Medical (38)	21%	13%	11%	11%	79%	Pathway; CSL Plasma; Sydney Health
Libraries/Demo (47)	19%	15%	0%	0%	81%	Cardboard; Samsung SmartTag; Addons for Melon
Business (39)	18%	15%	10%	10%	82%	Boat Browser; FedEx Mobile; Meta Business Suite
Communication (41)	17%	15%	0%	0%	83%	Messenger; Inbox Homescreen; Messenger – Texting App
Android-Wear (37)	14%	14%	8%	8%	86%	I am – Daily affirmations; CallApp: Caller ID&Block; Fitbod

*Different numbers of apps for each category (e.g., only six Finance apps) are due to DMCA, which prevented some APK/XAPK downloads.