Sweat, Sync, and Exposure: A Privacy Analysis of Interoperable Health and Fitness Ecosystems in Android

Aniketh Girish
IMDEA Networks Institute

Johanna T. Gunawan
Maastricht University

Srdjan Matic
University Carlos III of Madrid

Joel Reardon
University of Calgary

Juan Tapiador
University Carlos III of Madrid

Narseo Vallina-Rodriguez
IMDEA Networks Institute

Abstract

Health and fitness services on mobile platforms operate within interconnected ecosystems that aggregate sensitive data across wearable devices, mobile apps, cloud services, and third-party SDKs. Through integrations with services such as Google Fit, Fitbit, Strava, and Garmin Connect via OAuth 2.0, hundreds of apps can gain access to health records provided by other apps and devices, from heart rate to sleep patterns. Two access control mechanisms regulate health data at different levels: Android runtime permissions govern data on the device, while OAuth scopes govern data exposed through integration platforms. However, neither mechanism constrains how data is subsequently shared with third parties. Once accessed, health data and user identifiers can flow to other apps and to dozens of embedded third-party SDKs.

This paper presents the first large-scale empirical study of such health-data dissemination across Android health ecosystems. We study 1,548 health apps and 11 integration platforms using an analysis pipeline that combines static and dynamic analysis, BLE device simulation, sensor fuzzing, and network traffic instrumentation to capture cross-app data flows mediated by OAuth. Our analysis shows that 42.6% of analyzed apps integrate with at least one integration platform and that 57.2% transmit sensitive health data alongside persistent IDs to third parties, enabling cross-app profiling and re-identification for secondary purposes. Finally, we uncover structural privacy risks in OAuth-authorized integrations: 32% of integrating apps forward platform-sourced records to third-party SDKs, with little or no user awareness.

Keywords

Privacy, Online Tracking, e-Health, Android, OAuth, SDKs

1 Introduction

Health and fitness apps on Android collect and process some of the most sensitive data available on mobile devices. This includes, for example, heart rate, blood oxygen saturation, sleep stages, menstrual cycles, and body composition metrics. Even seemingly routine health measurements such as menstrual cycle logs or blood pressure readings can reveal underlying medical conditions [14].

As a result, regulators increasingly classify the collection and sharing of such data as high risk. The EU General Data Protection Regulation (GDPR) treats health and biometric data as special-category data under Article 9 [16], typically requiring explicit consent for processing. In the United States, the Federal Trade Commission (FTC) has enforced the Health Breach Notification Rule against apps that share health records alongside device identifiers [21, 22]. These concerns are not hypothetical: the FTC fined BetterHelp $7.8 million for sharing health data with advertising platforms, while consumer watchdogs have documented data sales by health apps to data brokers and health insurers [41, 69].

These regulatory concerns become particularly important in modern health ecosystems, where data increasingly flows through centralized integration platforms rather than remaining confined within a single app boundary. Modern integration platforms such as Google Fit, Fitbit, Strava, and Garmin Connect aggregate data from wearable devices, phone sensors, and third-party apps into cloud-hosted repositories and expose standardized APIs through which authorized third-party apps can access this data [31, 50, 64]. For example, a wearable companion app may upload years of heart rate readings, sleep records, and workout history to Garmin Connect, from which unrelated wellness, nutrition, or social fitness apps can independently retrieve that data without interacting with the original device or companion app.

Most platforms authorize this cross-app access through OAuth 2.0 [37]. When an app requests health data access, users are redirected to a web-based consent screen to approve platform-defined scopes (e.g., sleep, heart rate, or body measurements). The platform then issues access tokens that apps store locally and use to query the platform's REST-based APIs. Because consent happens in the browser and the app manages the tokens, users cannot easily monitor or revoke this access afterward. The exchange also occurs outside Android's permission system [29], which cannot enforce the granted scopes, list active grants, or revoke them. As access and refresh tokens may persist indefinitely [37], apps can continue retrieving health data long after initial authorization.

Once retrieved, platform-sourced health data enters the app process, where it becomes accessible to all embedded third-party code, including analytics and advertising SDKs due to the weak isolation that Android enforces between first- and third-party code [28, 57]. These SDKs can then observe the data through shared memory, logging, or network interception, without separate authorization or disclosure. These two mechanisms decide who accesses health data, but neither governs how it propagates across apps and SDKs.

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–19

https://doi.org/XXXXXXX.XXXXXXX

Despite the sensitivity of such health data and repeated regulatory enforcement, prior work has largely studied health apps in isolation: analyzing app permissions and network traffic [9, 18, 54, 72], reverse-engineering BLE protocols of individual devices [8, 24, 25], or identifying SDK presence through static analysis of WearOS apps [65, 70]. However, these app-centric approaches cannot capture ecosystem-level privacy risks arising from integration platforms where multiple apps and SDKs interact: apps retrieving cross-app health data via OAuth-authorized APIs, embedded SDKs observing that data within the shared process, or single apps aggregating records across multiple health platforms simultaneously. To the best of our knowledge, no prior study has systematically analyzed how these cross-platform and cross-app data flows create privacy risks in Android health apps at scale. To fill this gap, we analyze 1,548 health apps and 11 integration platforms on Google Play, addressing these research questions:

- **RQ1:** What health data, identifiers, and behavioral signals do Android health apps and their third-party SDKs collect, and how can these flows enable cross-app profiling?
- **RQ2:** How do OAuth-based APIs mediate cross-app and cross-platform health-data flows, and what privacy risks emerge from these integrations outside Android's permission model?

To answer these questions, we develop an analysis pipeline combining static SDK detection, dynamic analysis with BLE device simulation and sensor fuzzing, and network traffic instrumentation to systematically observe data collection, API interactions, and third-party SDK behaviors as data flows across apps, platforms, and their embedded SDKs. This work opens a new avenue to empirically study data flows beyond traditional process-centric approaches [5, 15]. Based on this analysis pipeline, we make the following key contributions:

(1) We provide the first large-scale characterization of Android health-data integration ecosystems (§6), analyzing 1,548 health and fitness apps and 11 integration platforms. We show that 660 of 1,548 apps (42.6%) communicate with at least one integration platform and that 53 of 55 possible platform pairs are interconnected through app-mediated or direct platform integrations. Our analysis reveals that health-data access is authorized locally, but propagated ecosystem-wide across apps, platforms, and backend services, creating complex distributed data-sharing paths that neither Android nor any individual platform can fully observe, audit, or control.

(2) We perform a large-scale empirical analysis of health-data dissemination across Android health ecosystems (§7), showing that 973 of 1,548 apps (62.9%) transmit sensitive health data and that 885 of 1,548 (57.2%) disseminate both health data and user or device IDs to third parties. We uncover widespread sharing of body-composition, workout, reproductive-health, and medical-condition data with advertising, analytics, and attribution platforms such as Meta, TikTok, AppsFlyer, and Mixpanel. Our analysis further reveals practices prohibited by platform policies, including ID bridging between resettable and persistent identifiers, plaintext transmission of email addresses, and the linkage of health data with precise location and behavioral telemetry, enabling durable cross-app profiling and re-identification, even after users reset their advertising IDs.

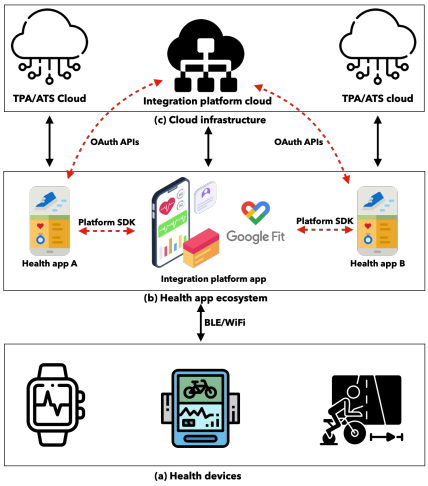

Figure 1: Health ecosystem data flow. (a) Smartwatches or cycling computers connect to mobile apps via BLE. (b) Apps exchange data with integration platforms through OAuth and health SDKs, enabling cross-app data consolidation. (c) Apps and platforms sync data with backend servers. Red dashed lines indicate data flows to third-party tracking SDKs.

(3) We uncover structural privacy risks introduced by OAuth-based health-data integrations (§8). By tracing platform API fields, OAuth credentials, and synchronized health records across apps, SDKs, and backend services, we find that, of the 76 apps accessing integration-platform data, 24 (32%) forward health-related signals to third-party services and SDKs, and 24 (32%) expose live OAuth credentials or authorization artifacts beyond the authorized client. Our findings show that OAuth authorization constrains which data an app may retrieve, but not how retrieved data and credentials propagate across embedded SDKs, analytics infrastructures, and cross-platform integrations outside Android's permission and visibility mechanisms.

More broadly, our results expose a fundamental limitation of app-centric privacy controls and analysis methods. Data flows that appear local to individual apps propagate across platforms, SDKs, and backend services that cannot be observed or governed in isolation, so capturing them requires dynamic, ecosystem-level analysis across app, platform, and network boundaries. We discuss potential mitigations in §9, including recommendations for platform-level enforcement, SDK sandboxing, and transparency mechanisms.

Responsible Disclosure: We shared a preprint of this paper with Google, as the operator of the Android platform, and with several European national data protection authorities. Google reviewed our report and determined that the reported behaviors are not a security vulnerability, while noting they may be considered for remediation in a future release.

2 Background

This section overviews health app types and their role (§2.1), communication paradigms in health mobile app ecosystems (§2.2), and access control mechanisms governing these flows (§2.3).

2.1 Health App Types

We define a *health-data-processing app* as any app published under the Google Play Health & Fitness category that collects or processes at least one of the 22 categories in our health-data taxonomy (§4). As we scope apps by the data they process rather than their category, a nutrition journal that logs diet and weight, a mindfulness app that tracks mood and stress, and a cycling app that records heart rate and routes all qualify. An app that gathers no data itself through Android APIs also qualifies when it reads a user's records from an integration platform, as it still processes health data. These apps fall into three subcategories that differ in how they collect data and interact with other apps, and in their role within this ecosystem:

- *Companion Apps* are developed by device manufacturers to pair with proprietary hardware via Bluetooth Low Energy (BLE) [24, 65, 70]. These apps collect biometric data from paired devices (heart rate, sleep stages), synchronize devices with cloud services, and manage user-introduced data.
- *Generic Health Apps* are hardware-independent apps that rely on smartphone-supported sensors (e.g., GPS) and user-introduced data (e.g., weight) to track health and activity [39, 72]. Some may support connections to third-party fitness devices.
- *Integration Platforms* are centralized data repositories that store health records gathered from multiple sources [31, 50, 64]. Unlike companion apps, they aggregate health data from multiple devices, third-party apps, and sensors into a unified store, typically through OAuth APIs, creating many-to-many data flows.

Regardless of their subcategory, any of these applications may integrate third-party software development kits (SDKs), some of which may be used for advertising and tracking purposes.

2.2 Communication Paradigms

Figure 1 shows how apps and integration platforms ingest health data from various sources before uploading it to cloud services. Each stage relies on a distinct communication channel.

Device-to-App. Wearable health devices communicate with mobile apps via short-range protocols like Ant+ [4] and BLE (Figure 1a) [6]. BLE relies on the Generic Attribute Profile (GATT), which organizes measurements into standardized services and characteristics identified by standardized UUIDs. The Bluetooth SIG defines standard profiles for common health metrics such as heart rate and running speed, but vendors may extend these with proprietary profiles. We consider BLE as the *de-facto* standard for device-to-app communications in smart health solutions.

App-to-Cloud. Companion and health apps upload health data over HTTPS (Figure 1c) to vendor-owned or self-hosted backends. Apps also embed *third-party SDKs*, i.e., prepackaged libraries for analytics, advertising, and crash reporting that run inside the app process. Because Android does not separate first- and third-party code within an app, these SDKs inherit the host app's privileges and data access and can collect and retransmit health data to their own backends (Figure 1c) [19, 40, 56].

Platform-Mediated Data Exchange. Apps interact with integration platforms through platform SDKs or direct REST calls that handle authentication and API communication (Figure 1b). A platform provides these SDKs to reach its own API, unlike the third-party SDKs above that apps embed for analytics and advertising. Platforms expose structured health records—activities, sleep sessions, heart rate series, body measurements—through REST APIs that support fine-grained queries over data type, date range, and detail level. For example, `GET /user/-/activities/heart/date/today/1d.json` on Fitbit's API retrieves one day of heart rate data. Through such APIs, apps both contribute and retrieve records originating from other apps or devices. Before apps can access this data, the user must explicitly connect their platform account to the app and authorize a set of data access scopes that determine which health data categories the app may access, typically using OAuth 2.0.

2.3 Access Control

Access to health data on Android is governed by multiple mechanisms, each controlling a different segment of the data flow path.

Runtime Permissions. The Android permission system regulates access to on-device sensors and data [29]. Several permissions are central to health apps: `ACTIVITY_RECOGNITION` (Android 10+) controls access to the Activity Recognition Transition API, allowing apps to infer physical activity types like walking or cycling from motion sensors without accelerometer access; `BODY_SENSORS` gates access to heart rate and other wearable physiological sensors; `BLUETOOTH_SCAN` and `BLUETOOTH_CONNECT` (Android 12+) gate BLE device discovery and connection; and `ACCESS_FINE_LOCATION` gates GPS-based activity tracking. Health Connect, Android's on-device health data store, extends this model with fine-grained permissions per data type (*e.g.,* `READ_HEART_RATE`, `READ_SLEEP`), each requiring independent user approval. All runtime permissions are enforced by the OS, visible to users, and revocable at any time.

OAuth 2.0 Authorization. OAuth 2.0 [37] allows third-party apps to obtain delegated access to user data hosted on remote services and platforms without requiring user credentials. In health ecosystems, platforms expose this delegated access as account linking: when a user connects a third-party app to their platform account (*e.g.,* linking a nutrition tracker to their Fitbit account), the user is redirected to a browser-based consent screen to review the requested data access scopes, then grant or deny access. Scopes are platform-defined permissions that specify which health data categories the app may read or write (*e.g.,* `heartrate.read`, `activity.read_all`). Upon approval, the platform issues an access token and, optionally, a refresh token, which the app stores locally and uses to query the platform's RESTful APIs. Refresh tokens let apps renew access without re-prompting users. This entire authorization flow, including consent, token management, and API access, operates outside Android's permission mechanisms.

These mechanisms operate independently and without mutual awareness. Android's runtime permissions cover on-device sensor and data-store access but not OAuth-authorized cloud API flows, and each integration platform enforces only its own scopes, with no visibility into the other platforms or embedded SDKs an app combines. We examine the resulting privacy risks in §3.

Table 1: Health data taxonomy. *Cat.* is the number of categories in each domain.

Domain	Cat.	Example Patterns
Physical Activity	4	steps, calories, distance, workout, pace, pedometer
Body Metrics	3	weight, bmi, body_fat, body_temp, respiration, muscle_mass
Cardiovascular	3	heartrate, bpm, spo2, systolic, diastolic, bloodpressure
Sleep & Recovery	3	sleep_stage, stress, readiness, hrv_score, mood
Medical Info	2	allergy, medication, diagnosis, health_risk, glp1
Nutrition & Hydration	2	nutrition, diet, protein, water_intake, hydration, meal_plan
Demographics	1	gender, age_range, biological_sex
Mental Wellness	1	meditation, mindfulness, breathing_exercise
Metabolic Health	1	blood_glucose, a1c, hba1c, blood_sugar
Weight	1	weight_loss_goal, target_weight, obesity
Women's Health	1	menstrual, ovulation, fertility, pregnant, nursing

3 Ecosystem Privacy Risks

The communication channels and authorization mechanisms described in §2 enable ecosystem-level health data flows invisible to Android's permission model. These flows create privacy risks related to identifier co-collection, cross-app data exchange, and fragmented authorization boundaries. We next describe the main exposure channels underlying each risk, which arise from either structural pitfalls or intentional misuse.

Side-Channels. The lack of isolation between first- and third-party code on Android allows SDKs within health apps to access data beyond their intended scope, as they inherit the execution context and privileges of the host app. For example, apps commonly use crash reporting and SDKs that collect custom logs, event traces, and debugging information [46, 57]. When health-related values are included in telemetry pipelines—e.g., heart rate or workout metadata—the data also becomes accessible to embedded SDKs. Exposure may also occur unintentionally when health data propagates through logging frameworks, serialization pipelines, or shared data structures monitored automatically by these SDKs (§2.3). Even the presence of a health-related log entry, independent of its contents, can reveal that a user engages with a sensitive category [46].

Identifier Bridging. Apps and their embedded SDKs collect multiple categories of identifiers alongside health data: mobile resettable advertisement identifiers (Android Advertising ID/AAID), persistent identifiers (Android ID and BLE MAC addresses), and derived identifiers (email addresses, account IDs) [28, 57]. These identifiers serve different purposes and provide varying levels of user control. Privacy risks escalate when entities co-collect identifiers across categories: an SDK that collects both the advertising ID and a user's email address can transform a resettable ID into a persistent one mapped to a real-world identity. As the same SDKs are embedded in many health apps, an SDK can further correlate health and behavioral signals across otherwise unrelated contexts, linking persistent identifiers to health data in ways no individual app intended.

Authorization Gap. Once health data is inside an app, no mechanism effectively governs where it flows next. Android permissions gate on-device sensors and OAuth scopes gate platform retrieval, but neither constrains how the data moves across app and SDK boundaries afterward. The fragmented access control framework described in §2.3 isolates platform-level authorization from process-level data access. Integration platforms authorize apps to access specific health data categories through scoped OAuth (§2.3); the app's embedded SDKs inherit that access without the platform's knowledge or the user's consent. Sensitive data therefore moves from the platform API, which enforces scoped access, into the shared app process, where embedded third-party code runs without equivalent isolation. An app holding tokens from multiple platforms may even concentrate data from independent authorization domains in one process, hence observable to every embedded SDK.

4 Dataset and Experimental Setup

Our dataset defines the scope of the study in four parts: the health data we look for, the apps and integration platforms we test, and the physical devices that drive them.

Health Data Taxonomy. We ground our health-data taxonomy in the regulatory definitions of sensitive health information under GDPR Article 9 [16] and the FTC Health Breach Notification Rule [20]. We cover 22 app categories grouped into 11 domains, from cardiovascular and metabolic signals to reproductive and mental-health data, each paired with the keyword patterns we use to detect it (Table 1). We apply these patterns to select candidate health apps (§4) and to flag health fields in captured traffic (§5).

App Selection. We scope our study to the health-data-processing apps defined in §2.1: those that collect or process at least one taxonomy category. Because it is self-declared and genre-based, the Play Store Health & Fitness category does not reliably indicate whether an app processes health data. We apply a multi-stage filter to retain qualifying apps. We crawl the Play Store using 150 health-related seed queries and expand results through store recommendations, yielding 35,000 candidate apps. We refine the crawled set into the *health-data-processing apps* defined in §2.1 with a two-stage, model-based filter, the only model-based stage in our pipeline. The first stage embeds each app description with a BERT sentence transformer (`all-mpnet-base-v2`) [58], reduces its dimensionality with UMAP [48], clusters the embeddings with HDBSCAN [38], and keeps the clusters tied to health monitoring, fitness tracking, and sensor-based activity logging. The second retains apps whose description exceeds a cosine-similarity threshold of 0.6 against a TF-IDF representation of our taxonomy keywords. To validate the filter, the lead author labeled 200 randomly sampled selected apps as relevant or not to our taxonomy; 188 (94%) were relevant, giving 94% precision over the retained set. The 12 false positives are predominantly non-fitness tracker utilities whose descriptions overlap our health vocabulary, such as shipment trackers (17TRACK), mileage loggers (MileIQ), and to-do apps (Productive). Residual errors cluster on overloaded terms that occur in both health and non-health contexts: activity (social feeds), stress (relief games), and cycle (motorcycle racing, billing cycles). The resulting dataset contains 2,348 apps spanning wearable companions, sport trackers, nutrition and wellness apps, and other health-related ones.

Integration Platform Selection. We identify integration platforms using two signals: references in app descriptions and confirmation through observed API communication during dynamic testing. We restrict our analysis to platforms that expose APIs or OAuth-based authorization flows enabling third-party access to user health

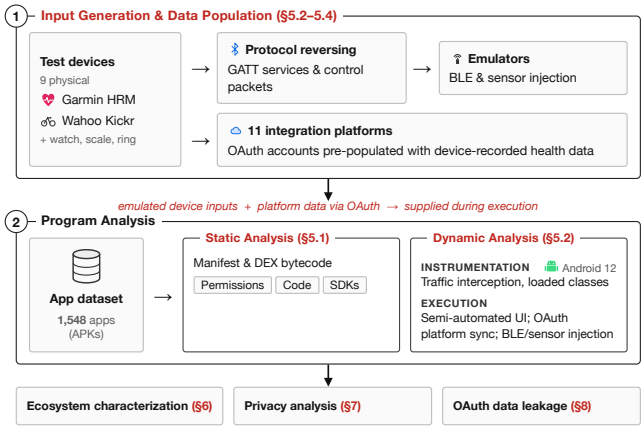

Figure 2: Methodology overview.

data. We identify 11 integration platforms in our dataset spanning both vendor-specific ecosystems (e.g., Fitbit, Garmin, Withings) and general-purpose platforms (e.g., Google Fit, Samsung Health).

Device Selection. We use real devices both to exercise app-device synchronization workflows and to dynamically test integration platform accounts with realistic health data as input. Our testing environment includes 9 physical devices—smartwatches, heart rate monitors, blood pressure monitors, smart cycling trainers, smart rings, and smart scales—listed by type, brand, and model in Table 7 (Appendix B.1). Each device connects to its companion app via BLE and syncs data to the corresponding integration platform, producing ground-truth records spanning cardiovascular, activity, sleep, and body-composition data.

5 Methodology

Answering our research questions requires observing health data flows across apps through the three channels used by modern health apps: BLE-connected devices, on-device sensors, and OAuth-authorized integration platforms [65, 70]. Observing all these data flows requires complementary analysis techniques. We combine static (§5.1) and dynamic analysis (§5.2) to capture both health data collection and dissemination across apps and integration platforms.

However, a key challenge is that many health apps are event-driven: synchronization, analytics logging, and platform API interactions are triggered only after the app receives the inputs from a paired device or local sensors. Simply launching the app therefore fails to exercise large portions of its health data processing pipeline. To generate realistic inputs at scale, we reverse engineer representative BLE fitness devices (§5.3) and use the recovered protocols to build BLE and sensor emulators (§5.4) that continuously generate health input data during app dynamic analysis.

5.1 Static Analysis

Static analysis identifies how each app acquires health data and determines necessary inputs for dynamic testing (§5.2). We analyze manifest declarations and DEX bytecode to identify health-data acquisition mechanisms and SDK dependencies.

Permission Analysis. We parse each app's `AndroidManifest.xml` to extract declared health-relevant permissions. BLE connectivity is signaled by `BLUETOOTH_SCAN`/`BLUETOOTH_CONNECT` (Android 12+); sensor access by `ACTIVITY_RECOGNITION` (Android 10+), `BODY_SENSORS`, and `ACCESS_FINE_LOCATION` for GPS-based activity tracking. For platform-integrated apps, we also look for OAuth redirect URI schemes indicating an embedded integration platform SDK.

Code Analysis. We use Androguard [11] to decompile APK bytecode and identify the mechanisms through which apps acquire health data. This analysis extracts three classes of signals:

(1) *BLE-based.* Companion apps communicate with peripheral fitness hardware by implementing `BluetoothGattCallback` and exchanging data over GATT characteristics. We scan apps for GATT service UUID constants matching known health profiles—Heart Rate Service (`0x180D`), Cycling Power (`0x1818`), Fitness Machine Service (`0x1826`)—and `BluetoothGattCallback` subclasses with characteristic handler call locations (`onCharacteristicRead()`, `onCharacteristicChanged()`). We route apps with BLE-based signals to BLE device emulation (§5.3).
(2) *Sensor-based.* Apps register listeners against Android's `SensorManager` to gather health data from the phone's on-board sensors. We find `SensorManager.registerListener()` call invocations with health-relevant sensor types (*e.g.,* `TYPE_STEP_COUNTER`, `TYPE_ACCELEROMETER`) in 22% of apps. We inject synthetic sensor input to test these apps.
(3) *Platform integration.* Apps use OAuth APIs to retrieve health records from integration platforms. We locate OAuth client initialization call locations, redirect handlers, REST API endpoint references, and integration-platform SDK signatures. We use pre-populated accounts to test apps with these behaviors (§5.2).

SDK Analysis. We use code-level SDK signatures derived from public SDK documentation, package registries, and prior work [17, 28, 47] to identify popular advertising and tracking SDKs (*e.g.,* Firebase and TikTok Ads). We extend it with a curated set of signatures for health integration SDKs (*e.g.,* Google Fit, and Garmin Connect), which aggregate health records through OAuth-based authorization flows. In total, our SDK detection draws on 759 code signatures.

5.2 Dynamic Analysis

Dynamic analysis captures how health data propagates across apps, aggregators, and third parties at runtime. Because the flows we study occur beyond the sandbox boundaries, conventional app-centric runtime analysis methods would observe only a fraction of them (§5.5). Of the 2,348 selected apps (§4), we simultaneously test 1,548 apps simultaneously, and in different combinations, on an instrumented Android 12 AOSP build supplied with realistic BLE and sensor measurements from our emulation framework (§5.4), excluding apps removed from the Play Store, incompatible with our build, or unable to launch. Our pipeline instruments the device to capture traffic, matches captured fields against health data and identifiers, and attributes each flow to the responsible SDK; we then detail the account setup and semi-automated execution driving it.

Instrumentation. We use a comercial well-tested instrumented AOSP build that monitors permission-protected API accesses and outbound network traffic at the OS level without modifying app behavior [1, 27, 28, 56, 57, 59, 67]. It intercepts TLS at the socket layer, recovering plaintext traffic even in certificate-pinned apps, and decodes common encodings (gzip, base64) before parsing. To capture code that static analysis misses, we instrument the Android Runtime, logging classes loaded at runtime through the class linker's `FindClass` method. We screen-record all manual tests to capture authorization flows and consent-interface behavior (§8.3).

Health-Data and Identifier Matching. We identify health data transmitted alongside user or device identifiers in the captured traffic through deterministic matching. From each flow, we extract field names and values from JSON bodies, event names, and HTTP headers, discard structurally uninformative fields such as protocol metadata and crash-reporting artifacts, and match the remainder against our 22-category taxonomy (§4) and the identifier types in Table 5. Health categories match on fixed key–value patterns, and identifiers match against the pseudonymous values seeded into each account and device, so identifier detection needs no separate validation. We label each matched field by destination—first-party, third-party, or integration platform—to attribute every disclosure to a recipient. Because we apply the matcher across hundreds of thousands of flows across thousands of apps, we favor precision over recall: a low false-positive rate keeps spurious detections rare at scale, and every reported flow is therefore a lower bound. On 400 manually labeled flows, the health-data matcher attains 0.93 precision and 0.86 recall across the three groups (Appendix A.2); residual errors stem from overloaded tokens like *height*, *period*, and *temperature*, which field-context rules remove.

SDK Attribution. We attribute each observed flow to a specific SDK by combining the static signatures of §5.1 with runtime class loading and the flow's destination endpoint and socket-level traces. Analytics and platform SDKs often obfuscate their code, encrypting endpoint URLs and mangling class names through ProGuard/R8. We therefore recover field names with SDK-specific heuristics and resolve obfuscated classes from the runtime class-loading trace, which static signatures alone miss. Because black-box testing lacks complete ground truth for in-app SDK behavior, and we do not observe in-process memory transfers between an app and its SDKs, we treat this attribution as best-effort.

Account and Device Setup. We pre-configure each testing device with unique pseudonymous identifiers (phone number, email address, username) to bypass app registration requirements and trace identifier dissemination in captured traffic. We create accounts on all 11 integration platforms using the same pseudonymous identifiers and pre-populate each account with real health data recorded by a researcher using the physical test devices (§4), tailored to each platform's data model. We populate activity platforms like Garmin Connect with workouts and GPS traces, and health platforms such as Fitbit and Withings with sleep and body-composition records. Because OAuth-connected apps retrieve pre-existing platform-hosted data, account pre-population is necessary to trigger the API flows observed in our analysis.

Semi-Automated App Execution. We test each app in a two-phase 10-minute session. Prior research showed that most tracking flows occur within the first few minutes of execution [59]. Note that fully automated execution is insufficient for this type of program: Android Monkey cannot navigate OAuth authorization flows, bypass registration screens, or locate integration settings buried within app menus. Therefore, during the first 5 minutes, a researcher manually interacts with each app to complete setup tasks such as onboarding, account registration, device pairing, and OAuth authorization using the pre-established platform accounts. For platform-integrated apps, the researcher navigates to the integration settings and completes OAuth authorization for all supported platforms through Chrome, enabling attribution of subsequent API requests to the corresponding authorization flow. Once authorization completes, the app immediately queries the integration platform's health data endpoints via REST; our instrumentation captures both the inbound responses—revealing which health data categories the app retrieves from each platform—and any outbound transmissions to the app's own servers or embedded third-party SDKs. The app is automatically tested for 5 minutes using the Android Monkey and the BLE peripheral emulator and sensor injector from §5.4 for network and sensor inputs, maintaining health data flows throughout and exercising additional UI paths.

5.3 BLE Reverse Engineering

Many health apps rely on external BLE devices and do not exercise their synchronization and analytics functionality unless valid device measurements are received [8, 24]. We observe that 73% of the apps in our dataset reference BLE APIs in their code, and to generate realistic device inputs at scale, we reverse engineer representative BLE protocols and reconstruct the message formats required for emulation. We analyze two devices spanning the protocol spectrum present in our corpus: (i) the Garmin HRM, which relies exclusively on Bluetooth SIG-standardized GATT profiles, and (ii) the Wahoo Kickr smart trainer, which combines standardized profiles with proprietary extensions. These two devices are not meant to be exhaustive. Using Frida [55] on a rooted Pixel 3a running Android 12, we instrument `BluetoothGattCallback` methods (Table 8) to capture characteristic reads, writes, and notifications exchanged between each device and its companion app. The recovered message formats are subsequently used to construct the BLE emulators described in §5.4. Detailed protocol specifications and packet structures are provided in Appendix B.2.

The Garmin Heart Rate Monitor implements the Bluetooth SIG Heart Rate Profile (HRP, UUID `0x2A37`) and transmits heart rate and RR-interval measurements to derive heart-rate variability metrics. As the protocol follows the published Bluetooth SIG specification, we decode packets to reconstruct the standard message format.

The Wahoo Kickr Smart Cycling Trainer combines Bluetooth SIG-standardized cycling profiles with a proprietary BLE service. Reverse engineering revealed that this custom service acts as the primary bidirectional control channel, carrying commands that configure resistance, automated ergometer resistance targets and road simulation parameters. We reconstructed the protocol by correlating intercepted command payloads with observable trainer

responses from hundreds of manual interactions, recovering a message format absent from any public specification (Appendix B.2). The resulting specification enables faithful emulation of trainer behavior during dynamic testing without human involvement.

5.4 Device and Sensor Emulation

The reconstructed BLE protocols provide the message templates and state machines needed to emulate realistic sensor inputs and continuously stimulate event-driven health apps during dynamic analysis without physical hardware. During static analysis we found that apps validate incoming sensor data and reject values outside expected physiological ranges. Both emulators therefore generate device-plausible signals rather than synthetic canaries.

BLE Device Emulation. Companion apps pair with an external BLE peripheral at startup and will not proceed past connection screens without one. To exercise these apps at scale without physical hardware, we implement a BLE peripheral emulator that advertises itself as a virtual fitness device and reproduces the GATT services recovered during protocol analysis, using `bleno` [49], a Node.js GATT server library deployed on a Raspberry Pi. Our emulator implements all profiles and message templates obtained from reverse engineering, including standard and proprietary BLE services. Generated sensor data and notification timings are constrained to realistic ranges observed in the corresponding physical devices (*e.g.,* heart rate 40–200 bpm), including GATT notification timing consistent with real hardware. Supported services and characteristics are listed in Table 9 in the Appendix.

On-Device Sensor Emulation. Sensor-based apps receive no BLE traffic and thus cannot be exercised by the BLE emulator alone. For these apps, we hook `android.hardware.SensorEventListener` and `onSensorChanged()` (via Frida) to inject synthetic accelerometer and gyroscope events at runtime, emulating activities such as walking, running, and cycling. All injected values model realistic fitness activity patterns bounded within physiological limits.

5.5 Limitations

As with any large-scale black-box-based privacy analysis, our methodology has inherent scope limitations, making our results a lower bound on ecosystem-wide health-data collection.

- **Platform coverage.** Our aim is ecosystem-level propagation rather than exhaustive per-app characterization. We accordingly exercise each app's integration flows to the extent manual interaction permits, completing multi-factor authentication and free-tier account setup where feasible. Integrations gated behind paid subscriptions or account-specific restrictions therefore fall outside our study, so our results are a lower bound.
- **BLE protocol coverage.** Our protocol reconstruction focuses on two representative devices: the Garmin HRM and the Wahoo Kickr. Although these devices span the two protocol classes observed in our dataset, other vendor-specific protocols may expose behaviors not captured by our emulation framework.
- **Input generation coverage.** BLE and sensor emulation target apps built on Android's standard GATT and sensor frameworks. Apps using proprietary protocols, vendor-specific libraries, or undocumented hardware interfaces fall outside this scope.
- **Behavioral coverage.** We focused on functionalities exercised during onboarding, account setup, OAuth authorization, synchronization, and active app usage. Behaviors triggered only after prolonged use, premium feature activation, geographic restrictions, or highly specific user interactions may not be observed.
- **Manual interaction.** Completing onboarding, registration, device pairing, and OAuth authorization requires a human in the loop, as automated UI drivers such as Android Monkey cannot navigate these flows. This bounds the scale of our analysis. LLM-driven UI agents could in principle automate these steps, but doing so at our scale would incur substantial computational, economic, and carbon costs, which we leave to future work.

6 Integration Platform Ecosystem

Our research questions require understanding the software ecosystem through which health data flows, including how apps obtain health data and the OAuth-based mechanisms integration platforms use to mediate cross-application data sharing (Figure 1b). Because apps often integrate multiple platforms and inherit access to combined datasets, the exposure surface extends beyond any individual application or platform. Thus, the Android health integration ecosystem defines the data flows, trust boundaries, and authorization relationships underlying the privacy risks we analyze in §7 (SDK dissemination) and §8 (OAuth leakage).

6.1 Health Data Integration Models

Among the 1,548 dynamically tested applications, 833 (53.8%) reference at least one integration platform in their app description, and 660 (42.6%) actively communicate with one or more health-data integration platforms. These platforms govern access through OAuth delegation or OS-managed permissions, exposing health records through category-specific access controls. Table 2 summarizes the capabilities and authorization models of our 11 platforms.

Data Sources. In our experiments, all 11 platforms ingest data from third-party applications, while most (9/11) also support vendor devices and some (4/11) integrate phone sensors. These differences create a heterogeneous ecosystem for aggregating health records from devices, phone sensors, and third-party services.

Authorization Mechanisms. All studied platforms expose a common abstraction: apps request access to specific health-data types and receive credentials that permit subsequent retrieval or synchronization of records. Nine platforms rely on OAuth authorization, while Samsung Health supports both OAuth and Android permissions, and Health Connect relies exclusively on native permission controls. OAuth authorization occurs outside Android's native permission model, allowing platforms to grant access independently of OS-level enforcement, as shown in §8.

Access Control Granularity. OAuth-based access is controlled through *scopes*, i.e., permissions that define which health records an application may access. As shown in Table 2, most health platforms expose fine-grained scopes for specific data categories (e.g., heart-rate samples or sleep sessions), while several additionally provide aggregate summaries or bulk historical exports. Identity-related scopes (e.g., profile and email access) are also common; among platforms where scope requests were observable, roughly one third of integrating apps request user or device IDs alongside health data.

Table 2: Characterization of health integration platforms by data source, authorization mechanisms, and access types.

Platform	% Apps[1]	% Apps[2]	#Scopes[3]	Data Source		Authorization		Access Granularity			R/W Ops	
	descr.	traff.		Native Sensors	Third-party Device	OAuth	Perms	Fine	Coarse	Bulk	R	W
Fitbit	34.05%	27.71%	20		✓	✓		✓	✓	✓	✓	✓
Garmin Connect	24.03%	11.50%	5		✓	✓		✓	✓	✓	✓	✓
Google Fit	24.03%	15.83%	9	✓	✓	✓		✓	✓	✓	✓	✓
Google Health Connect	1.55%	10.21%	–	✓			✓	✓	✓		✓	✓
myfitnessPal	20.87%	4.72%	2			✓			✓	✓	✓	✓
Oura	8.46%	2.58%	3		✓	✓		✓			✓	✓
Polar Flow	13.37%	4.13%	2		✓	✓		✓			✓	
Samsung Health	2.46%	1.23%	6	✓	✓		✓	✓	✓	✓	✓	✓
Strava	19.06%	6.46%	7	✓	✓	✓		✓		✓	✓	✓
Suunto	2.00%	1.36%	0		✓	✓		✓	✓		✓	–
Withings	25.00%	4.07%	3		✓	✓		✓	✓		✓	

[1] Share of the 1,548 analyzed apps whose Play-Store description references the platform. [2] Share of apps whose runtime traffic contains requests to the platform's API. [3] Number of distinct OAuth scopes documented on the platform ("–" indicates permission-based platforms that expose no OAuth scopes).

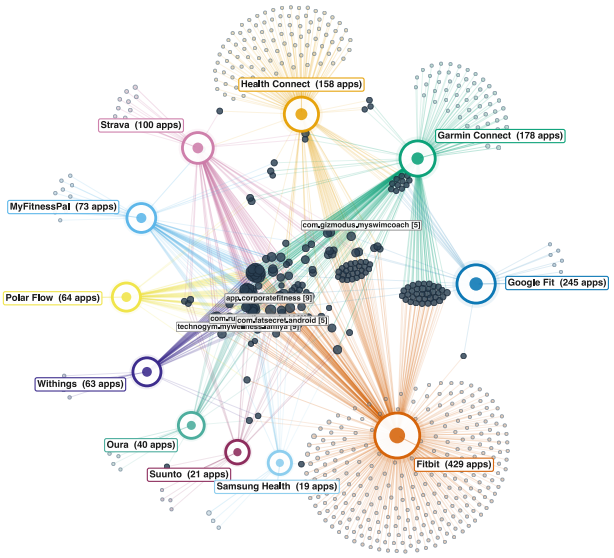

Figure 3: Network graph of 1,548 health apps (gray) and 11 integration platforms (colored). Gray edges connect platforms sharing common apps. Node size denotes the number of connected apps. Labels identify selected highly connected apps, with bracketed numbers indicating platform count.

Read and Write Operations: Most platforms (8/11) support both data retrieval and data contribution through separate authorization scopes. Consequently, applications integrating multiple platforms can act as intermediaries to read records from one service and write to another. Only the intermediary can observe the entire end-to-end data flow: the source platform observes only the read operation and the destination platform only the write operation, leaving the end-to-end transfer invisible to either platform.

6.2 Multi-Platform Integration

Although integration platforms authorize access independently (§6.1), applications frequently combine multiple platforms within the same execution context. Consequently, health data can traverse platform boundaries through both app-mediated synchronization and direct platform-to-platform integrations. Across the 55 possible platform pairs studied, 53 are connected by at least one observed integration, indicating a highly interconnected ecosystem.

App-Mediated Integration. Figure 3 maps the dynamically observed app–platform relationships. Most applications integrate a single platform (460 of 660 integrating apps, 69.7%), while the remaining 200 (30.3%) connect two or more platforms and therefore act as potential bridges between otherwise independent authorization domains. Integration concentrates around a small number of dominant platforms. Fitbit appears in the traffic of 429 of 1,548 apps (27.71%), followed by Google Fit (245, 15.83%) and Garmin Connect (178, 11.50%). Fitbit and Google Fit, the two most integrated platforms, are both operated by Google. While most multi-platform apps integrate only a small number of services, a small tail of highly connected applications draws on many platforms at once, forming large aggregation points for health data from diverse sources. The breadth of data these integrations carry varies sharply. Cronometer, a nutrition tracker, synchronizes only body metrics such as height and weight from Garmin and Polar, whereas `app.corporatefitness`, a corporate-wellness app, aggregates body composition, demographics, and menstrual-cycle and blood-oxygen tracking settings across Fitbit, Garmin Connect, Oura, and Polar into a single profile. Integrations also carry health data beyond the fitness context: FitnessBank (`com.affinitybank`) integrates Fitbit, Garmin Connect, and Oura to derive a customer's deposit interest rate from their step count.

Direct Platform Integration. Figure 4 captures direct integrations between platforms that operate without third-party application involvement. To construct this graph, we manually trigger each integration through in-app interactions and validate the observed edges against the platform's public integration documentation and captured traffic. Of the observed integrations, 40% occur through on-device Android APIs (IPC-based, in red) and 60% through OAuth-based REST interfaces (in blue). These relationships concentrate around a small number of aggregation hubs. Health Connect receives on-device writes from seven other platforms, while Strava receives more inbound OAuth integrations than any other service. Conversely, MyFitnessPal is the largest OAuth source, exporting data to six platforms. These integrations allow a single measurement to propagate across multiple services automatically depending

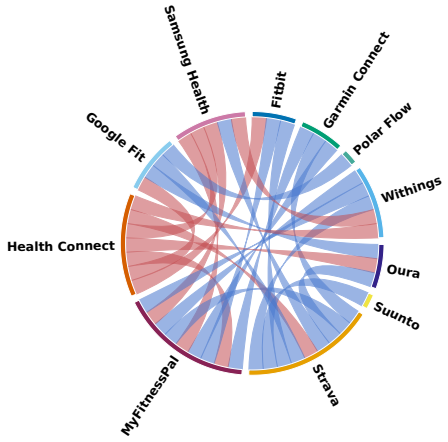

Figure 4: Direct platform integrations. Red chords denote on-device app-level integrations; blue chords denote OAuth-based REST integrations between platform backends. Separated from Figure 3 for readability.

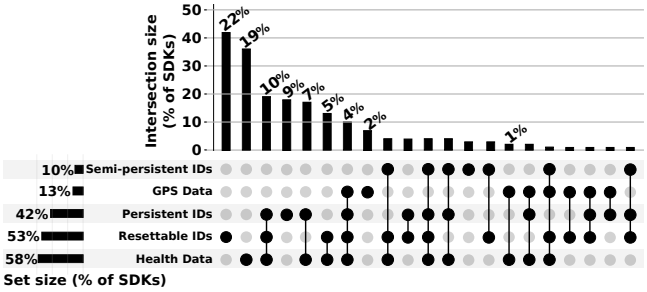

Figure 5: Co-collection of health, IDs, and location data across 193 SDKs collecting at least one signal. Left bars give each category's prevalence among all SDKs; top bars give the size of each co-collection patterns. For example, 112 of 193 (58%) SDKs receive health data; of these 112, 76 (68%) also collect at least one ID—the sum of all intersection bars whose dot pattern includes both the Health row and at least one ID row.

on user actions. For example, a reading generated by a Withings device may subsequently appear in Samsung Health, Health Connect, MyFitnessPal, and Strava without further user interaction.

> **Takeaway #1.** Each integration platform authorizes an app independently and sees only the requests made to it, yet the ecosystem is densely interconnected: apps draw on several platforms at once, and platforms sync directly with one another, so a single health record can spread across many services from one user action. Health Connect manages local on-device permissions, but not the inter-platform OAuth or server-to-server links that carry data between them.

7 Privacy Analysis

This section examines how the ecosystem relationships identified in §6 translate into concrete data dissemination. We first quantify the dissemination of health data and identifiers to third-party services (Figure 1c), showing how analytics, advertising, and attribution SDKs receive health data linked to persistent identifiers and, in some cases, bridge resettable and persistent identities (§7.1). We then present tracking pixels and tag managers as a case study of these flows (§7.2), and analyze unintended dissemination channels, where health data propagates to third-party SDKs through developer-defined analytics events and telemetry mechanisms (§7.3). Table 10 in Appendix C lists all the package names and install counts associated with the apps mentioned in this section.

7.1 Health Data Dissemination

Health data is routinely transmitted to third parties alongside persistent identifiers that enable cross-app profiling and re-identification. Using the dynamic analysis pipeline described in §5, we study runtime data flows across 1,548 apps; Appendix A defines the health categories and identifier types we detect. We find that 1,298 of 1,548 apps (84%) transmit identifiers, 973 of 1,548 (62.9%) transmit health data, and 885 of 1,548 (57.2%) transmit both. Together, these apps contact 4,711 unique domains, of which 2,450 (52%) belong to third parties offering advertising and tracking services. Body-composition data (weight and BMI) is the most commonly transmitted category (400 of 1,548 apps, 25.8%), followed by activity and workout information (137/1,548, 8.9%), demographic attributes (100/1,548, 6.5%), and reproductive health data such as menstrual cycle information (91/1,548, 5.9%).

This data is overwhelmingly sent to general-purpose analytics, advertising, and profiling services rather than health-specific providers. Figure 5 maps how health, identifier, and location collection overlap across the 193 third-party SDKs that collect at least one of these signals. Of these, 112 (58%) receive health data; every percentage in the remainder of this subsection is computed over that recipient subset. The recipients include widely deployed analytics services (Amplitude, Firebase Analytics, Mixpanel), the customer-data platform Segment, and the ad network Google DoubleClick—none health-specific, yet each receives enough to infer a user's physical condition, habits, and lifestyle.

Of these health-receiving SDKs, 76 of 112 (68%) also collect at least one identifier, enabling health records to be linked to persistent identities and cross-app profiles. These identifiers include the resettable Android Advertising ID (AAID), email addresses, and persistent device identifiers such as the Android ID (§3). Table 5 lists the full set we search for. More concerning, 33 of 112 health-receiving SDKs (29%)—e.g., the advertising SDKs AppLovin, Vungle, and Flurry—simultaneously collect both the resettable AAID and a persistent identifier. This bridging (§3) nullifies the AAID reset, enabling cross-app tracking, a practice Google Play policies prohibit [32]. Notably, 445 of 732 leaked email transmissions (60.8%) are in plaintext, while most of the remainder use reversible encodings such as Base64 or URL encoding rather than hashing. Where advertising and attribution SDKs such as TikTok, Meta, Adjust, and AppsFlyer receive SHA-256 digests, the hashes still enable cross-service identity linkage [23].

We observe health data being disseminated alongside both GPS coordinates and router-derived signals such as WiFi SSIDs and BSSIDs, which act as location proxies [28] and add an additional layer of sensitivity [10]. Of the SDKs that receive health data, 15 of

112 (13%) also collect location information, and 10 of 112 (9%) combine location with both resettable and persistent identifiers. When combined with health data, these leaks reveal sensitive contexts such as a user's residence, workplace, exercise habits, and visits to healthcare facilities. Google therefore classifies precise location as sensitive and prohibits linking it to personal data [32].

7.2 Case Study: Tracking Pixels

Tracking pixels and tag managers exemplify how health information and identifiers are combined in practice [61, 67]. In our dataset, 832 of 1,548 apps (53.7%) load at least one pixel, 37 (2.4%) transmit health data to one, and 26 (1.7%) send both health data and identifiers to the same service. Meta Pixel (755 apps, 48.8%) is the most prevalent, followed by Google Tag Manager (193 apps, 12.5%) and TikTok Pixel (76 apps, 4.9%). Across these integrations, body composition, activity, demographic, and reproductive health data are routinely transmitted together with hashed email addresses or advertising identifiers. In the Appendix, Tables 11 and 12 list the pixel and tag-manager services observed in our dataset and the apps contacting six or more of them. We attribute all pixel activity to embedded WebViews, which handle onboarding, account creation, and OAuth flows. Two examples follow:

Meta Pixel. Among the 755 apps that load Meta's web conversion pixel, 333 (44.1%) share user IDs, and 21 (2.8%) leak both health data and identifiers by transmitting hashed email addresses in the `ud[external_id]` field alongside the persistent `_fbp` cookie. We observe workout habits, mental and body metrics being transmitted through the pixel, transforming health-related onboarding and assessment workflows into advertising and profiling signals without explicit visibility to users. The Noom app, for example, embeds a pixel in its onboarding survey, reporting each response to Meta together with coarse location, the persistent `_fbp` cookie, and a hashed user identifier (`ud[external_id]`). In many cases, Meta also collects the same data through its native SDK.

TikTok Pixel. Among the 76 apps communicating with TikTok services, 25 (32.9%) transmit identifiers and 4 (5.3%) disseminate health data, including demographic attributes, menstrual-cycle information, medical conditions, weight-management goals, and body-composition metrics. These data flows are linked through TikTok identifiers and SHA-256 hashed email addresses, enabling the construction of persistent user profiles. For example, the Noom app also embeds TikTok Pixel on its onboarding questionnaire. We observe events disclosing a user's medical conditions, pregnancy status, gender, and weight loss goals, allowing TikTok to reconstruct responses across an entire health assessment workflow.

7.3 Custom Events and Crash Telemetry

Health data also reaches third-party SDKs through general-purpose telemetry channels not designed to carry it. As §3 discusses, the authorization gap lets apps expose health data to embedded SDKs in two ways: developers log it into generic analytics and attribution events, and crash-reporting SDKs capture it incidentally when logging execution traces for debugging.

Custom-Event Logging. Many analytics, attribution, and advertising SDKs expose generic event APIs for key-value pairs. Apps use these to transmit body-composition, workout, reproductive-health, mood, medical-condition, and medication data to analytics services (Amplitude, Mixpanel) and attribution SDKs (Singular, AppsFlyer), typically linked to an email or the Android Advertising ID. Figure 6 shows a representative example from the MapMyRide app, which packages workout distance, duration, heart rate, pace, device identifiers, and location information into a custom analytics event sent to Amplitude. MapMyRide's privacy policy permits sharing health data with analytics providers but does not specify what it shares or how [52]. ASICS RunKeeper presents the same behavior through an `Activity Saved` event. Similar patterns appear across health, fitness, wellness, and medical apps as shown in Table 3.

```
"event_type": "performance_monitor",
"device_info": {
  "androidADID": "XXXX-XXXX-XXXX-XXXX",
  "manufacturer": "Google", "device_model": "AOSP",
  "gps_enabled": true,
  "location": { "lat": "XX.XX", "lng": "-X.XX" },
},
"event_properties": {
  "activity_type": "Road Cycling", "pedometer": true,
  "workout_distance": X, "workout_duration": X,
  "workout_heart_rate": X, "workout_speed": X,
}
```

Figure 6: A custom event sent by MapMyRide to Amplitude with workout metrics, location coordinates, the AAID, and contextual metadata. Sensitive fields are highlighted.

Table 3: Representative examples of sensitive data disclosures through SDK logging custom events (CE) and telemetry (T).

App	Data Type	SDK	ID	Log.
CareClinic	Symptoms, medication log, sleep, mood	Mixpanel	Email	CE
EatMyRide	Nutrition and fitness logs	Sentry	User Prof.	T
LetsGetChecked	Medical test interests	AppsFlyer	AAID	CE
MapMyRide	Workout metrics, heart rate, GPS	Amplitude	AAID	CE
Oura	HRV, sleep, stress scores	Crashlytics	User Prof.	T

Telemetry and Crash-Reporting SDKs. Crash-reporting and performance-monitoring SDKs often collect console logs, request traces, and app states by default to support debugging. When developers log health-related data, the same information becomes part of telemetry reports transmitted to these parties. We observe this behavior across Sentry, Firebase Crashlytics, and Datadog SDKs. For example, the EatMyRide app logs backend responses containing body metrics, demographic attributes, VO_2max measurements, heart-rate zones, and daily nutrition records, which Sentry subsequently captures and includes in error reports sent to the vendor. Similarly, the telemetry for the YAZIO app exposes weight, BMI, age, and sex through debugging information collected by the same service. For example, we observe Crashlytics reports containing user-profile data, sleep, stress, and heart-rate data, and Datadog monitoring workout summaries, calories, and heart-rate samples.

Although many of these SDKs act as service providers for the developer, they still receive sensitive health data, and because downstream processing is not observable from the client side, we cannot tell whether it is retained, combined, or repurposed.

Takeaway #2. Health data disseminates from these apps to the third-party analytics and advertising SDKs they embed, tied to persistent identifiers that several of them bridge across apps. Whatever permission gated the data's entry, none governs this onward flow, and the embedded SDKs inherit the app's access (the authorization gap of §3). The recipients are general-purpose analytics and advertising services rather than health providers, so sensitive health signals reach them potentially without the user's awareness.

8 Data Leakage via OAuth

Once a user grants an OAuth connection to an integration platform, the app obtains a token, calls the platform's API, and receives health records in return. From there, those records are transmitted to embedded SDKs and linked to external IDs without additional authorization, visibility, or platform oversight. This section follows that path end to end, from the platform's API response (Figure 1b), through the app's bytecode, to a third-party SDK (Figure 1c). To trace platform-sourced data at scale, we crawl the API documentation of the 10 integration platforms with a public web API and label every documented response field as a health-data category, an identifier, or an OAuth credential. We then search captured traffic for these fields with the same deterministic key-value matching as our health-data detection (§5.2), attributing each transmission to the process that emitted it. To avoid spurious hits, we match distinct values such as OAuth tokens and activity IDs exactly, while common measurements such as a heart rate or weight must also agree on their surrounding fields. This strict rule makes every reported flow a lower bound. Within a single session, 76 apps in our dataset forward what they pull from a platform to a third-party service. A matched value need not be a raw measurement, as even a configuration setting or consent flag discloses what a user tracks or has agreed to share. We document the platform health data itself (§8.1), the OAuth credentials that authorize access to it (§8.2), and usability and transparency in OAuth-based integrations (§8.3).

8.1 Platform Health Data in Third-Party Traffic

Apps that authorize an integration platform routinely feed what they retrieve to already-embedded third-party SDKs. For example, 24 of 76 apps (32%) forward a health-revealing signal to a third party in the same session, and 42% of those forward one in a special category under the GDPR (Article 9). Across these 24 apps, which often forward several signal types at once, health-revealing metadata (19) and event labels (16) are the most common signals, ahead of OAuth artifacts and account identifiers (16), raw measurements (9), and configuration or consent fields (7). The recipients are usually general-purpose analytics and customer-data platforms, such as Amplitude and Segment, and not health providers. Worse, because 34 of 76 apps (45%) draw on two or more platforms (§6.2), the signal that leaves carries no trace of which platform it came from.

Body weight, activity type, and heart rate leave as raw values but the most sensitive categories do not. Menstrual cycle, blood glucose, and blood oxygen readings usually stay within first-party code, and what reaches third parties is health-revealing *metadata* (Table 4) in the form of flags indicating that the user tracks a domain (`cycleInsightsEnabled`) or an event type naming a synced record (`healthCareType=HEART`). Yet even a bare flag may reveal the entire condition. Moreover, these records are not anonymous.

Table 4: Representative examples of health data forwarded to third parties by apps that pull from an integration platform.

App	Platforms	Third parties	Health signal
Corporate Fitn.	Fitbit, Garmin, Polar	Segment, Braze, RudderStack	Menstrual-cycle, blood-oxygen, period-prediction
RunMotion	Fitbit, Garmin, Polar	Segment	Menstrual-cycle, blood-oxygen flags
RunKeeper	Garmin, Polar	Amplitude	Heart-rate
Bearable	Fitbit, Google Fit	Mixpanel, Segment	Heart rate, body temp, weight

```
"event": "GoogleFitRestingHeartRateCount",
"$email": "<user-email>",
"properties": {
  "healthCareType": "HEART"/"BODY_TEMP"/"WEIGHT",
  "enabledWidgets": ["MOOD","SYMPTOM","SLEEP","MEDICATION
      "],
  "numActiveSymptoms": 2, "numActiveSymptomGroups": 2,
  "numberOfHeartRateEntries": 2,
  "screen": "UserConfig.HealthSyncConfig" }
```

Figure 7: A single Bearable event received by both Mixpanel and Segment; highlighted fields are sensitive.

The same SDKs sit inside many apps, so one SDK collects health data across all of them, and Segment alone receives data drawn from seven platforms. In 12 of 24 chain apps (50%), these records arrive alongside the user's email. The Bearable event in Figure 7 exemplifies this end-to-end chain: it pulls heart rate, sleep, and blood pressure from Fitbit and Google Fit, then forwards the category of each synced record, the conditions the user tracks, and how many symptoms and heart-rate entries they log to Mixpanel and Segment in the same event as the email. What lands at the third party is not a stray health reading but a cross-platform health profile tied to a single, identifiable user.

8.2 Credential and Identifier Leakage

OAuth artifacts leak from apps in three forms: credentials that let a recipient act on the platform as the user, account identifiers that link the user's platform account to third-party profiles, and OAuth flow metadata (scopes, client IDs, state parameters, redirect URIs) that exposes what the app has authorized. We report each in turn.

Credentials. Of the 76 integrators, 24 (32%) expose at least one live OAuth token[1] to code beyond the authorized client (an embedded SDK or an ungranted back-end). All 24 apps expose a refresh token; 22 of them also expose an access token. A further 2 of 76 apps (3%) hold valid tokens for more than one platform at once (e.g., MyFitnessCoach, Mindbody), so any embedded SDK gains visibility into relationships spanning platforms the user may perceive as independent. Beyond in-app exposure, Four apps (5%) forward platform credentials to their own first-party backend—outside the flow the platform authorized—and four apps forward them to third-party trackers (e.g., myCols and sevenbold send a live Strava

[1]An access token authorizes API requests on the user's behalf for a limited time; a refresh token is a longer-lived credential used to mint fresh access tokens. A valid exposed token lets its recipient read the user's platform data directly, as the user, without re-authorization.

access token to the ad network Adzerk at `engine.adzerk.net`; Cadence sends its Strava authorization code to the attribution SDK Branch, which is directly redeemable at Strava's `/token` endpoint for live tokens per RFC 6749 §4.1 [37]). Whoever receives such a credential can query the platform API as the authenticated user; a refresh token outlives the session until the user manually revokes it. Google's OAuth policies require tokens to remain with the authorized client [30], and Google Play's User Data Policy prohibits transferring personal or sensitive data to third parties without user consent [32]; the flows above are difficult to reconcile with either.

Account Identifiers. Of the 76 apps, 9 (12%) transmit platform-specific identifiers such as a Strava athlete ID or a Garmin login to a non-platform host. Unlike an advertising ID, which carries no meaning outside the ad ecosystem, a platform identifier is durable and identifies a real user account. A Strava athlete ID, for instance, resolves directly to the user's public page at Strava's web endpoint. When such an identifier arrives at a tracker (e.g., embedded in the app context of Sentry crash reports), the tracker gains a stable link between the user's wearable identity and whatever advertising or analytics profile it already maintains for that user. Google Play's User Data Policy classifies persistent identifiers as personal data and prohibits their linkage with sensitive categories, including health, without an appropriate legal basis [32].

OAuth Metadata. A third class of leak concerns the metadata of the OAuth flow itself—scopes, client identifiers, redirect URIs, and `state` values. 3 of 76 apps (4%) transmit such metadata to analytics, advertising, and attribution SDKs, often through the tracking pixels loaded into the consent flow (§7.2). For example, RunMotion sends Polar `scope`, `state`, and `redirect_uri` values to Google Analytics, and Trenara sends Polar and Strava `client_id`, `state`, and `redirect_uri` to Branch, Snapchat, and Google Analytics. These parameters reveal which platforms a user connects to and the permissions they granted, let third parties correlate OAuth events with pre-existing advertising and analytics identifiers, and—in the case of a `client_id`—fingerprint the app's registered OAuth client. The `state` value is an OAuth 2.0 [37] CSRF token scoped to the authorized client (§10.12); its appearance at a tracker subverts that scope and is sanctioned by none of the platforms we observe.

These disclosures differ in impact. An exposed identifier or OAuth parameter cannot read the user's data but reveals which platforms they use and links their wearable identity to a tracker's profile. An exposed token—or an authorization code exchangeable for one—is more consequential: its recipient can read the user's account as the user, until the app is uninstalled or access revoked.

8.3 Potential User Experience Influences

Third-party-to-integration-platform connections are inherently privacy-sensitive, but the user interaction flows for setting them up may further contribute to users' potential exposure. To investigate the potential for future work on dark patterns in OAuth interactions, and to supplement findings in §8.2 and §8.1, we conduct cursory qualitative analyses [60] on a subset of screen recordings captured from our dataset. We inspected the integration platforms' authorization flows using recordings taken during dynamic testing. Second, we examined a subset of third-party health app recordings, documenting recurring interface behaviors that nudged or constrained user decisions around data access. Further detail on our pilot analysis is provided in Appendix C.1.

These pilot inspections yielded samples of the preselection (Garmin, Strava, Suunto), visual hierarchy (Fitbit Mobile and Google Fit), and nagging (PolarFlow) dark patterns, consistent with prior consent flow work [34, 51, 62], as well as variance in integration platforms' choice architecture decisions in OAuth flows. In particular, in third-party app integrations, Fitbit provided granular and non-preselected permissions controls to users (as comparatively "better" practices), while we noted other integration platforms' permissions flows commonly preselected all possible options for health data (Figure 9). This variance in UI decisions even for one common protocol highlights the potential for undue influence originating from the design layer, atop otherwise standardized mechanisms.

We also note the potential for upstream data leakage issues. In the case of MyFitnessPal, while no immediate dark pattern concerns arose in our brief examination of OAuth interactions, we saw the integration platform requiring consent to transfer data to the US in onboarding interactions, albeit with opt-in, fine-grained checkbox controls (Figure 10). This may have implications for "onward transfers" under Art. 44 of the GDPR to organizations not certified with the EU-US Data Privacy Framework. As a first party, MyFitnessPal appears not to be EU-US DPF certified, and discloses the potential transfer to other uncertified parties. Thus the lawfulness of MyFitnessPal's data consent mechanism at registration may yet be in violation of the GDPR, warranting future research regarding onward transfers for integration platform data.

These observations emphasize the need for future work contextualizing OAuth flows within the user experience and privacy policies of integrating parties, and for investigating additional vectors from OS or browser-based consent mechanisms. This would add to broader scholarship on the dark pattern privacy implications of health-oriented mobile apps [2] and technical OAuth studies [12].

Takeaway #3. The authorization gap between Android permissions and platform OAuth (§3) leaves apps free to forward what they retrieve to the third-party SDKs they already embed, carrying the user's email or platform account but no trace of which platform the data came from, and leaving the user no way to control where it goes. Many also leak the OAuth artifacts that authorize the access, including a live token in 32% of them. The consent interfaces that grant this access frequently preselect broad scopes and obscure what is shared (§8.3).

9 Discussion

Health data enables personalized services and wellness insights, but when collected outside traditional medical settings it is treated as a special category under GDPR Art. 9 and the FTC Health Breach Notification Rule [16, 20]. Our results expose a structural cross-application authorization gap in the Android platform (§3): data and credentials granted to one app can propagate across process boundaries without user awareness, OS mediation, or platform oversight enabled by integration APIs like OAuth (§7, §8). Google Play policy already prohibits many of the observed behaviors, yet the abuse is difficult to detect because conventional analyses test apps in isolation, missing data flows that emerge only through interactions across apps and processes. Moreover, Google Play's Data Safety labels meant to disclose these practices to users are

self-reported and often inaccurate [1, 43, 57]. We next discuss its primary root causes and potential mitigations.

OAuth-Mediated Cross-Platform Exposure. OAuth-mediated integration is a structurally distinct exposure channel invisible to Android controls. The protocol scopes *who* may access platform data; it does not scope where the data goes next. Across the 11 platforms that we study (§6), consent screens differ in scope granularity, default preselection, and the language used to describe what is being authorized. Several apps split scope across nested screens or omit it from the user-facing copy entirely (§8.3)—producing a checkbox at the moment the privacy decision should be most legible. Of the 76 apps that integrate a platform from their own bytecode, OAuth artifacts leak beyond the authorized client in three forms: credentials (access and refresh tokens), account identifiers (login emails), and OAuth metadata (scopes, client IDs, and state parameters). Most consequentially, 32% of these apps expose a live access or refresh token, so the recipient can query the platform API as the user, and a refresh token outlives the session until the user manually revokes it. No runtime indicator exposes these flows.

SDK-Mediated Health-Data Dissemination. In addition to OAuth-based channels, the SDK layer provides a pathway for health data to propagate into the broader tracking ecosystem due to weak separation between first- and third-party code on Android. We identify 112 third-party SDKs receiving health data across our corpus, 68% of which link health data with at least one persistent or resettable identifier in the same session (§7.1). The bridge defeats Android's resettable-advertising-ID protection: once an AAID is co-collected with an email or login ID at the same SDK, resetting the AAID no longer breaks continuous tracking. The recipients are general-purpose analytics, advertising, and crash-reporting libraries—not specialized health vendors—inheriting access to physiological data through app instrumentation, not through health-specific opt-in. Under GDPR Art. 9, profiling using biometric or physiological signals requires explicit consent or another valid legal basis. This data dissemination warrants regulatory scrutiny.

Mitigations. Creating adequate boundaries in this ecosystem requires layered action across multiple stakeholders. Because OAuth grants have no runtime indicator or user-facing list of active platform grants (§8), *mobile platforms* should expose these flows the way they expose local sensor access—a runtime indicator, an inspectable log, and an inventory of active platform delegations—and strengthen isolation between first-party app code and embedded SDKs to limit incidental SDK access to platform-returned data [19]. *Integration platforms* should publish per-token audit trails, tighten scope semantics so identity, profile, and health cannot be granted on a single tap, and expose first-party revocation paths inside the originating app. Given the pre-selected broad scopes and coercive wording we observe in consent flows (§8.3), *app stores* should add automated detection of these OAuth consent patterns, complementing existing checks on runtime-permission misuse. Auditing cross-app propagation at scale, however, remains challenging: it requires runtime instrumentation across arbitrary program combinations, however, creates a combinatorial explosion that motivates future work on scalable cross-app analysis.

10 Related work

Security and Privacy in Health Ecosystems. Prior work has shown that health and fitness apps routinely transmit sensitive health metrics (*e.g.,* heart rate or activity data) to third parties, often alongside persistent identifiers and without meaningful user transparency [2, 24, 39, 63, 73]. Other studies examined cross-device privacy risks in wearable companion apps [65, 70] and vulnerabilities in BLE fitness devices and protocols, including spoofing, eavesdropping, and unauthorized device control [8, 25, 54]. While these efforts focus primarily on isolated apps, devices, or communication protocols, our work studies ecosystem-level privacy risks arising from OAuth-based integrations and cross-platform data sharing across apps, platforms, devices, and embedded third-party SDKs.

Security and Privacy of OAuth Integrations. A growing body of work studies the security and privacy of OAuth-based delegation. Dimova *et al.* [12] showed that OAuth-based websites routinely request more scopes than necessary for user identification. Other studies examine brokered single sign-on [42], cross-app OAuth attacks against integration platforms [45], and OAuth implementation flaws in Android apps [68]. These studies focus on the OAuth protocol itself, predominantly on the web; we complement them by characterizing how OAuth-mediated access reshapes the privacy surface of mobile health ecosystems, where retrieved data lands in a shared app process alongside embedded third-party SDKs.

Program Behavior Exploration. Prior dynamic-analysis systems have used UI automation and instrumentation frameworks to explore Android apps [3, 7, 44] and other smart-device ecosystems such as smart speakers, TVs, and voice assistants [13, 35, 66, 71]. In the BLE domain, tools such as Zwack [53], Oarsman [26], and Gymnasticon [36] emulate standard fitness-device profiles to test applications without physical hardware. However, existing systems are not designed to capture ecosystem-level health-data flows spanning multiple BLE devices, OAuth integrations, sensors, and third-party SDKs simultaneously. Our work builds on these foundations by combining BLE emulation, sensor fuzzing, and dynamic instrumentation to analyze cross-app data flows at scale.

11 Conclusions

This paper reveals significant privacy risks in the Android health ecosystem: health data propagates across devices, apps, and integration platforms, ultimately reaching advertising and tracking services alongside persistent identifiers. These findings expose a structural privacy gap driven by two architectural weaknesses: OAuth credentials extend access across app and platform boundaries, while the lack of isolation between first-party code and embedded SDKs allows the latter to inherit the host app's privileges and access sensitive data, including data retrieved via OAuth. These risks emerge from the composition of independently authorized components in user-space rather than from any single app, making them difficult to capture through app-level audits and existing platform transparency mechanisms. Protecting sensitive health data in multi-app ecosystems requires moving beyond app-centric controls toward platform-enforced isolation, cross-process auditing, and stronger accountability for data access.

Acknowledgments

We thank the reviewers and the shepherd for their valuable feedback and suggestions for improving our paper. This research was supported by the Spanish MCIN/AEI/10.13039/501100011033 through grants PID2022-140126OB-I00 (CYCAD) and RED2024-154240-T (EMACS), and by project PID2022-143304OB-I00 (PARASITE) funded by MICIU/AEI/10.13039/501100011033/ and by the ERDF, EU. This paper is co-funded by the European Union and the European Cybersecurity Competence Centre under grant agreement No. 101309318 (REAL-PETS). Narseo Vallina-Rodriguez has been appointed as a 2019 Ramón y Cajal fellow (RYC2020-030316-I) funded by MICIU/AEI/10.13039/501100011033. Srdjan Matic has been funded by the Madrid Regional Government under the César Nombela grant 2025-T1/COM-36349. This work was also supported by NSERC Discovery Grant RGPIN-2018-04237. The opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of any of the funding agencies.

Ethical Considerations

Our study analyzes the behavior of mobile applications and integration platforms and involves no human subjects. Dynamic analyses ran on test devices owned by the research team using synthetic accounts created at each integration platform, with no real user data collected or processed. The two BLE devices whose data flows we reverse-engineered (Garmin HRM-Dual, Wahoo Kickr) were purchased and instrumented by the team; and it was limited to the BLE communication between the device and its companion app to automatize app testing, and did not involve any firmware modifications or interactions with the device's internal hardware. Our testbed operates within each platform's published rate limits and never touches an account other than our own. OAuth credentials were stored securely, testing used only researcher-generated synthetic data, and no third parties received real user health data. We disclosed the credential and identifier leakage findings to the affected platforms, and kept the disclosures confidential for a time window aligned with industry practices. We also reported our findings to Google representatives due to the structural issues of this ecosystem, and to EU Data Protection Agencies.

Open Science

We release the curated list of 2,348 candidate apps, the 759 third-party SDK detection signatures, our BLE emulator for the 2 reverse-engineered devices, the data analysis scripts, and the processed datasets underlying the results in §7 and §8. The artifacts are available at https://github.com/Aniketh01/wearables-analysis.

AI Use

The authors used generative AI-based tools to revise the text, improve flow, and correct any typos, grammatical errors, and awkward phrasing. We have manually verified the integrity of this output. AI-based tools were not used anywhere else in this work. Specifically, we do not use AI-based tools as part of our research methodology, nor to generate figures. The literature review is of our own.

References

[1] Noura Alomar, Joel Reardon, Aniketh Girish, Narseo Vallina-Rodriguez, and Serge Egelman. 2025. The Effect of Platform Policies on App Privacy Compliance: A Study of Child-Directed Apps. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.

[2] Ghada Alsebayel, Giovanni M. Troiano, Johanna Gunawan, Herman Saksono, and Casper Harteveld. 2025. "An Ad Posing as Medical Advice": User Accounts of Dark UX in FemTech mHealth Apps. In *Proceedings of the ACM on Human Computer Interaction (HCI)*.

[3] Android. 2025. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey.

[4] ANT Wireless. 2003. ANT Wireless Protocol. https://www.thisisant.com/.

[5] 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. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[6] Bluetooth Special Interest Group. 2020. Connected Health: Powering a Healthier World with Bluetooth Technology. https://www.bluetooth.com/blog/connected-health-powering-a-healthier-world-with-bluetooth-technology/.

[7] Nataniel P. Borges Jr., Jenny Hotzkow, and Andreas Zeller. 2018. DroidMate-2: A Platform for Android Test Generation. In *International Conference on Automated Software Engineering (ASE)*.

[8] Marco Casagrande, Eleonora Losiouk, Mauro Conti, Mathias Payer, and Daniele Antonioli. 2022. BreakMi: Reversing, Exploiting and Fixing Xiaomi Fitness Tracking Ecosystem. In *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*.

[9] Britt Cyr, Webb Horn, Daniela Miao, and Michael A. Specter. 2014. Security Analysis of Wearable Fitness Devices (Fitbit). MIT 6.857 Computer and Network Security class project.

[10] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. 2013. Unique in the Crowd: The Privacy Bounds of Human Mobility. In *Scientific Reports*.

[11] Anthony Desnos and Androguard Contributors. 2025. Androguard: Reverse Engineering, Malware and Goodware Analysis of Android Applications. https://github.com/androguard/androguard.

[12] Yana Dimova, Tom Van Goethem, and Wouter Joosen. 2023. Everybody's Looking for SSOmething: A large-scale evaluation on the privacy of OAuth authentication on the web. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.

[13] Daniel J. Dubois, Roman Kolcun, Anna Maria Mandalari, Muhammad Talha Paracha, David R. Choffnes, and Hamed Haddadi. 2020. When Speakers Are All Ears: Characterizing Misactivations of IoT Smart Speakers. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.

[14] Electronic Frontier Foundation. 2022. Security and Privacy Tips for People Seeking Abortions. https://www.eff.org/deeplinks/2022/06/security-and-privacy-tips-people-seeking-abortions.

[15] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *USENIX Symposium on Operating Systems Design and Implementation*.

[16] European Parliament and Council of the European Union. 2018. General Data Protection Regulation (GDPR) Compliance Guidelines. https://gdpr.eu.

[17] Exodus Privacy. 2024. Exodus Privacy: Privacy Audit Platform for Android Applications. https://exodus-privacy.eu.org/en/.

[18] Kassem Fawaz, Kyu-Han Kim, and Kang G. Shin. 2016. Protecting Privacy of BLE Device Users. In *Proceedings of the USENIX Security Symposium*.

[19] Álvaro Feal, Julien Gamba, Narseo Vallina-Rodriguez, Primal Wijesekera, Joel Reardon, Serge Egelman, and Juan Tapiador. 2020. Don't Accept Candies from Strangers: An Analysis of Third-Party SDKs. In *Computers, Privacy and Data Protection Conference (CPDP)*.

[20] Federal Trade Commission. 2023. Complying with FTC's Health Breach Notification Rule. https://www.ftc.gov/business-guidance/resources/complying-ftcs-health-breach-notification-rule-0.

[21] Federal Trade Commission. 2023. FTC Enforcement Action to Bar GoodRx from Sharing Consumers' Sensitive Health Info with Advertising. https://www.ftc.gov/news-events/news/press-releases/2023/02/ftc-enforcement-action-bar-goodrx-sharing-consumers-sensitive-health-info-advertising.

[22] Federal Trade Commission. 2023. FTC to Ban BetterHelp from Revealing Consumers' Data Including Sensitive Mental Health Information to Facebook. https://www.ftc.gov/news-events/news/press-releases/2023/03/ftc-ban-betterhelp-revealing-consumers-data-including-sensitive-mental-health-information-facebook.

[23] Federal Trade Commission. 2024. No, hashing still doesn't make your data anonymous. https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2024/07/no-hashing-still-doesnt-make-your-data-anonymous.

[24] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. 2017. Breaking Fitness

Records without Moving: Reverse Engineering and Spoofing Fitbit. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.
[25] Hossein Fereidooni, Tommaso Frassetto, Markus Miettinen, Ahmad-Reza Sadeghi, and Mauro Conti. 2017. Fitness Trackers: Fit for Health but Unfit for Security and Privacy. In *IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*.
[26] Bruno Fernandez-Ruiz. 2020. Oarsman. https://github.com/olympum/oarsman.
[27] Aniketh Girish, Tianrui Hu, Vijay Prakash, Daniel J. Dubois, Srdjan Matic, Danny Yuxing Huang, Serge Egelman, Joel Reardon, Juan Tapiador, David Choffnes, and Narseo Vallina-Rodriguez. 2023. In the Room Where It Happens: Characterizing Local Communication and Threats in Smart Homes. In *Proceedings of the Internet Measurement Conference (IMC)*.
[28] Aniketh Girish, Joel Reardon, Juan Tapiador, Srdjan Matic, and Narseo Vallina-Rodriguez. 2025. Your Signal, Their Data: An Empirical Privacy Analysis of Wireless-scanning SDKs in Android. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.
[29] Google. 2024. Permissions on Android. https://developer.android.com/guide/topics/permissions/overview.
[30] Google. 2026. OAuth 2.0 Policies. https://developers.google.com/identity/protocols/oauth2/policies.
[31] Google Fit. 2025. Google Fit Help - What Data Does Google Fit Store? https://support.google.com/fit/answer/6098255?hl=en&co=GENIE.Platform%3DAndroid.
[32] Google Play. 2024. User Data Policy for Android Developers. https://support.google.com/googleplay/android-developer/answer/10144311?sjid=18338717510598425318-EU.
[33] Colin M. Gray, Michael L. Rivera, Robin S. Brewer, Elizabeth L. Churchill, Lorrie Faith Cranor, Allison McDonald, and Yixin Zou. 2024. An Ontology of Dark Patterns Knowledge. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*.
[34] Colin M. Gray, Cristiana Santos, Nataliia Bielova, Michael Toth, and Damian Clifford. 2021. Dark Patterns and the Legal Requirements of Consent Banners: An Interaction Criticism Perspective. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*.
[35] Zhixiu Guo, Zijin Lin, Pan Li, and Kai Chen. 2020. SkillExplorer: Understanding the Behavior of Skills in Large Scale. In *Proceedings of the USENIX Security Symposium*.
[36] Gymnasticon Contributors. 2020. Gymnasticon. https://github.com/ptx2/gymnasticon.
[37] Dick Hardt. 2012. The OAuth 2.0 Authorization Framework. RFC 6749, https://datatracker.ietf.org/doc/html/rfc6749.
[38] HDBSCAN Developers. 2022. How HDBSCAN Works. https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html.
[39] Andrew Hilts, Christopher Parsons, and Jeffrey Knockel. 2020. Every Step You Fake: A Comparative Analysis of Fitness Tracker Privacy and Security. https://citizenlab.ca/wp-content/uploads/2020/11/Every_Step_You_Fake.pdf.
[40] Gabriel Hortea, Aniketh Girish, Narseo Vallina-Rodriguez, and Juan Tapiador. 2026. Dead Domains, Living Data: A Privacy Risk Analysis of Domain Lifecycle in Android Apps. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.
[41] Christopher Ingraham. 2018. An Insurance Company Wants You to Hand Over Your Fitbit Data So They Can Make More Money. Should You? https://www.washingtonpost.com/business/2018/09/25/an-insurance-company-wants-you-hand-over-your-fitbit-data-so-they-can-make-more-money-should-you/.
[42] Tommaso Innocenti, Louis Jannett, Christian Mainka, Vladislav Mladenov, and Engin Kirda. 2025. "Only as Strong as the Weakest Link": On the Security of Brokered Single Sign-On on the Web. In *IEEE Symposium on Security and Privacy (S&P)*.
[43] Rishabh Khandelwal, Asmit Nayak, Paul Chung, and Kassem Fawaz. 2024. Unpacking Privacy Labels: A Measurement and Developer Perspective on Google's Data Safety Section. In *Proceedings of the USENIX Security Symposium*.
[44] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: A Lightweight UI-Guided Test Input Generator for Android. In *International Conference on Software Engineering (ICSE)*.
[45] Kaixuan Luo, Xianbo Wang, Pui Ho Adonis Fung, Wing Cheong Lau, and Julien Lecomte. 2025. Universal Cross-app Attacks: Exploiting and Securing OAuth 2.0 in Integration Platforms. In *Proceedings of the USENIX Security Symposium*.
[46] Allan Lyons, Julien Gamba, Austin Shawaga, Joel Reardon, Juan Tapiador, Serge Egelman, and Narseo Vallina-Rodríguez. 2023. Log: It's Big, It's Heavy, It's Filled with Personal Data! Measuring the Logging of Sensitive Information in the Android Ecosystem. In *Proceedings of the USENIX Security Symposium*.
[47] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. LibRadar: Fast and Accurate Detection of Third-Party Libraries in Android Apps. In *International Conference on Software Engineering (ICSE)*.
[48] Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. https://arxiv.org/abs/1802.03426.
[49] Sandeep Mistry. 2025. Bleno. https://github.com/noble/bleno.
[50] MyFitnessPal. 2025. What Kind of Products and Apps Work with MyFitnessPal? https://support.myfitnesspal.com/hc/en-us/articles/360032274232-What-kind-of-products-and-apps-work-with-MyFitnessPal.
[51] Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. 2020. Dark Patterns after the GDPR: Scraping Consent Pop-ups and Demonstrating their Influence. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*.
[52] Outside Interactive, Inc. 2023. Outside Inc. Privacy Policy. https://www.outsideinc.com/privacy-policy/?tab=custom_35e44d37-1739-4d5f-b0c4-be6525238b9d.
[53] Pedro Paixao. 2018. Zwack. https://github.com/paixaop/zwack.
[54] Mahmudur Rahman, Bogdan Carbunar, and Madhusudan Banik. 2013. Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device. In *Privacy Enhancing Technologies Symposium (PETS)*.
[55] Ole André V. Ravnås. 2021. Frida: Dynamic Instrumentation Toolkit. https://frida.re.
[56] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. 2018. Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile Tracking Ecosystem. In *Network and Distributed System Security Symposium (NDSS)*.
[57] Joel Reardon, Álvaro Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, and Serge Egelman. 2019. 50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System. In *Proceedings of the USENIX Security Symposium*.
[58] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
[59] Irwin Reyes, Primal Wijesekera, Joel Reardon, Amit Elazari Bar On, Abbas Razaghpanah, Narseo Vallina-Rodriguez, and Serge Egelman. 2018. "Won't Somebody Think of the Children?" Examining COPPA Compliance at Scale. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.
[60] Johnny Saldana. 2013. *The Coding Manual for Qualitative Researchers*. Sage Publishing.
[61] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik Zuiderveen Borgesius. 2022. Leaky Forms: A Study of Email and Password Exfiltration Before Form Submission. In *Proceedings of the USENIX Security Symposium*.
[62] Than Htut Soe, Oda Elise Nordberg, Frode Guribye, and Marija Slavkovik. 2020. Circumvention by Design – Dark Patterns in Cookie Consent for Online News Outlets. In *Nordic Conference on Human-Computer Interaction (NordiCHI)*.
[63] Animesh Srivastava, Sai Teja Peddinti, and Nina Taft. 2024. Unveiling Privacy Perspectives about Mobile Health Apps on a Large Scale. In *PETS Workshop on Privacy, Safety and Trust for Mobile Health Apps*.
[64] Strava Integrations. 2025. Strava Apps and Integrations. https://support.strava.com/hc/en-us/sections/203774187-Apps-Integrations.
[65] Marcos Tileria, Jorge Blasco, and Guillermo Suarez-Tangil. 2020. WearFlow: Expanding Information Flow Analysis To Companion Apps in Wear OS. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.
[66] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. 2020. The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.
[67] Tim Vlummens, Aniketh Girish, Nipuna Weerasekara, Frederik Zuiderveen Borgesius, Gunes Acar, and Narseo Vallina-Rodriguez. 2026. Bridges to Self: Silent Web-to-App Tracking on Mobile via Localhost. In *Proceedings of the USENIX Security Symposium*.
[68] Hui Wang, Yuanyuan Zhang, Juanru Li, Hui Liu, Wenbo Yang, Bodong Li, and Dawu Gu. 2015. Vulnerability Assessment of OAuth Implementations in Android Applications. In *Annual Computer Security Applications Conference (ACSAC)*.
[69] Shoshana Wodinsky and Kyle Barr. 2022. Data Brokers Are Selling Pregnant People's Data for a Post-Roe World. https://gizmodo.com/data-brokers-selling-pregnancy-roe-v-wade-abortion-1849148426.
[70] Doguhan Yeke, Muhammad Ibrahim, Güliz Seray Tuncay, Habiba Farrukh, Abdullah Imran, Antonio Bianchi, and Z. Berkay Celik. 2024. Wear's my Data? Understanding the Cross-Device Runtime Permission Model in Wearables. In *IEEE Symposium on Security and Privacy (S&P)*.
[71] Yangyong Zhang, Abner Mendoza, Guangliang Yang, Lei Xu, Phakpoom Chinpruthiwong, and Guofei Gu. 2019. Life after Speech Recognition: Fuzzing Semantic Misinterpretation for Voice Assistant Applications. In *Network and Distributed System Security Symposium (NDSS)*.
[72] Wei Zhou and Selwyn Piramuthu. 2014. Security/Privacy of Wearable Fitness Tracking IoT Devices. In *Iberian Conference on Information Systems and Technologies (CISTI)*.
[73] Noé Zufferey, Kavous Salehzadeh Niksirat, Mathias Humbert, and Kévin Huguenin. 2023. Revoked just now! Users Behaviors toward Fitness-Data Sharing with Third-Party Applications. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*.

A Dataset Supplement

This appendix supplements the dataset and methodology described in §4–§5. We document the device and user identifiers our pipeline detects in captured traffic (§A.1) and validate the health-data matcher against manually labeled flows (§A.2).

A.1 Device and User Identifiers

Table 5 lists the identifier and PII types we search for in captured traffic. Users cannot reset persistent identifiers; resettable ones can be cleared via a factory reset, a settings toggle, or a reinstall.

Table 5: Identifier and PII types considered in this study. *Pers.*: persistent (not user-resettable); *Reset*: resettable.

Identifier	Pers.	Reset	Definition
WiFi MAC	✓		Hardware ID of the WiFi interface.
HWID	✓		Hardware ID assigned by the manufacturer.
GSF ID	✓		ID tied to the device's Google account.
Email	✓		User's email address.
Android ID	✓		App-scoped ID; resets only on factory reset.
AAID		✓	Google advertising ID; resettable in settings.
Firebase ID		✓	App-instance ID; resets on uninstall or data clear.
Bluetooth Name		✓	User-defined, non-unique Bluetooth name.
Router MAC	✓		MAC address of the connected router.
Router Scan MAC	✓		Hardware addresses of nearby scanned routers.
Router Scan SSID	✓		Names of nearby scanned routers.
Router SSID	✓		Name of the connected router.
Device User Agent			Non-unique Attribute set (OS version, browser, …).
Coarse Geolocation		–	Approximate location from network data.
Fine Geolocation		–	Precise GPS-based location.

A.2 Health-Data Matcher Validation

We validate the health-data matcher (§5.2) on 400 manually labeled flows drawn from the 767,417 flows captured in the study. We sample these flows to cover both the matcher's positive decisions and the cases most likely to expose errors. The sample contains three groups. The first is 200 flows the matcher flagged as health data, spread evenly across the 22 categories so that the rarer but most sensitive ones, such as menstrual and medical data, appear in the sample rather than being crowded out by high-volume categories. The second is 150 near misses that mention health-adjacent terms but carry no health data. The third is 50 flows drawn at random and unrelated to health. For each flow, we determine by inspection whether it carries health data and compare this ground truth against the matcher's output.

The matcher reaches 0.93 precision and 0.86 recall across the three groups. Per-category precision (Table 6) ranges from 0.85 for menstrual health to 1.00 for blood oxygen.

The 14 flows incorrectly labeled as health data are dominated by a few overloaded tokens. *Height* matches screen dimensions, *period* matches subscription-billing fields, *temperature* matches device-battery readings, and *age* matches the HTTP `Age` header. For instance, `screen.height=2088` names a screen dimension and `fb_iap_subs_period=31` a billing period, both of which the bare pattern would otherwise tag as health data. These patterns are few and regular, so we apply a simple field-context regex over the surrounding keys and values that keeps a match only when its context is genuinely health-related, and drops it otherwise.

Table 6: Per-category precision of the health-data matcher on the labeled sample.

Category	Prec.	Category	Prec.
Blood oxygen	1.00	Activity distance	0.91
Sleep	0.97	Nutrition & diet	0.90
Heart rate	0.95	Stress & recovery	0.90
Body temperature	0.94	Medical conditions	0.89
Blood glucose	0.93	Body composition	0.88
Demographic	0.92	Menstrual health	0.85
Blood pressure	0.92		

B Device Reverse Engineering and Instrumentation

B.1 Test Devices and Instrumented APIs

Table 7 lists the 9 physical devices used to generate ground-truth health data and populate integration-platform accounts (§4). Table 8 lists the Android APIs we hook with Frida for BLE reverse engineering and sensor monitoring (§5.3).

Table 7: Physical test devices used in our dynamic-analysis pipeline, with vendor product pages.

Device Type	Brand	Model	Ref.
Smartwatch	Google	Pixel Watch 2	https://store.google.com/product/pixel_watch_2
Smartwatch	Samsung	Galaxy Watch 5	https://www.samsung.com/us/business/support/owners/product/galaxy-watch5-lte/
Smartwatch	Garmin	Venu	https://www.garmin.com/en-US/p/643260/
Smartwatch	Withings	ScanWatch	https://www.withings.com/us/en/scanwatch
Heart-rate monitor	Garmin	HRM-Dual	https://www.garmin.com/en-US/p/649059/
Smart cycling trainer	Wahoo	Kickr	https://www.wahoofitness.com/devices/indoor-cycling/bike-trainers/kickr-buy
Smart ring	Oura	Ring (Gen 3)	https://ouraring.com/product/rings/oura-gen3
Smart scale	Withings	Body+	https://www.withings.com/us/en/body-plus
Blood-pressure monitor	Withings	BPM Connect	https://www.withings.com/us/en/bpm-connect

Table 8: APIs instrumented via Frida for BLE reverse engineering and dynamic sensor monitoring.

Category	Hooked API	Purpose
Sensor	`SensorManager.registerListener()`	Detect sensor registrations
Sensor	`SensorEventListener.onSensorChanged()`	Capture real-time readings
BLE GATT	`onCharacteristicRead()`	Intercept BLE reads
BLE GATT	`onCharacteristicWrite()`	Capture write commands
BLE GATT	`onCharacteristicChanged()`	Log GATT notifications
BLE GATT	`getProperties()`	Decode characteristic properties
BLE GATT	`getUuid(), getService().getUuid()`	Resolve UUIDs
BLE GATT	`getValue()`	Extract payload bytes

B.2 BLE Protocol Details

This appendix details the BLE protocols we reconstructed for the 2 representative devices analyzed in §5.3: the Garmin HRM, which relies exclusively on Bluetooth SIG-standardized profiles, and the Wahoo Kickr, which combines standardized cycling profiles with a proprietary control service. The recovered message formats drive the device emulators of §5.4; the full set of GATT services and characteristics our emulator reproduces is listed in Table 9.

Heart Rate Profile (Garmin HRM). The Garmin HRM implements the SIG Heart Rate Profile (HRP). Each measurement packet begins with a *Flag byte* that specifies the data format and which optional metrics are present:

- *Heart-rate value format* (bit 0): selects whether the heart-rate value is encoded as an 8-bit (UINT8) or 16-bit (UINT16) integer, determining how the following bytes are interpreted.
- *Sensor-contact status* (bits 1–2): a 2-bit code reporting whether sensor-contact detection is supported and, if so, whether the device is currently in contact with the skin.
- *Energy expended* (bit 3): signals the presence of an Energy Expended field (in kilojoules) later in the packet.
- *RR-interval presence* (bit 4): marks the inclusion of one or more RR-interval measurements.
- *Reserved* (bits 5–7): reserved for future use.

The heart-rate value, in beats per minute, follows in the indicated format. RR-intervals—the time between successive heartbeats, from which heart-rate variability is derived—appear last, after the heart-rate and Energy Expended fields. Each is transmitted as a little-endian UINT16 formed from a byte pair and divided by 1024 to convert to seconds. All fields ride on the standard Heart Rate Measurement characteristic (UUID `0x2A37`), so a SIG-compliant parser recovers each one directly: we decode the stream by following the published field order, with no reverse-engineering effort.

Standard Cycling Profiles (Wahoo Kickr). The Wahoo Kickr is more involved than the HRM. Where the HRM exposes a single standardized characteristic, the trainer splits its telemetry and control across two SIG-defined cycling profiles and a proprietary service (described next). Of the standardized profiles, the Cycling Speed and Cadence Profile (CSCP) reports cycling kinematics from which speed and distance are derived:

- *Wheel revolution data*: cumulative wheel revolutions since reset, paired with the timestamp of the last detected wheel event.
- *Crank revolution data*: cumulative crank revolutions with the last crank-event timestamp, used to compute cadence.

Beyond kinematics, the Fitness Machine Service (FTMS) streams real-time training data and accepts control commands:

- *Indoor Bike Data*: instantaneous power (watts), resistance level, speed, and cadence, reported continuously during a workout.
- *Supported ranges*: the trainer's resistance-level and power ranges, advertised so the app can constrain its control commands.
- *Fitness Machine Control Point*: a writable characteristic through which the companion app adjusts resistance and target power and drives the bidirectional control loop.

Custom Cycling Power Service (Wahoo Kickr). The proprietary control service introduced in §5.3 (UUID `A026E005-0A7D-4AB3-97FA-F1500F9FEB8B`) carries a compact binary command format that we reconstruct in Figure 8. Every frame is little-endian (LSB first) and has a fixed layout: a one-byte *status* field, a one-byte *opcode* that selects the operation, a short field we could not attribute, a two-byte *target value* holding the setpoint, and a trailing *data* field. The opcode alone covers the trainer's full set of control operations. An *unlock* command first gates the channel; the app then issues direct setpoints—*resistance*, *power* (the ERG target), or *standard mode*—or enters *simulation mode*, which models outdoor riding from five physical parameters: road *slope* (grade), the rolling-resistance coefficient (*CRR*), *wind resistance*, *wind speed*, and *wheel circumference*. Because the trainer acknowledges each write and streams the resulting state back, we pair every opcode with its observable effect—resistance steps, ERG engagement, simulation-mode changes—and fix the field layout to the byte.

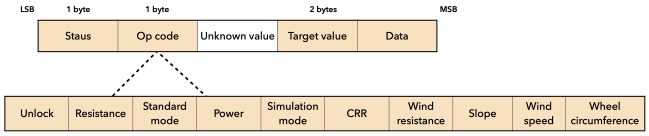

Figure 8: Decoded frame format of the Wahoo Kickr's custom Cycling Power Service (UUID `A026E005-0A7D-4AB3-97FA-F1500F9FEB8B`), reconstructed via BLE reverse engineering (§5.3). The 1-byte opcode (expanded below) selects the operation; none of these commands are defined in any Bluetooth SIG specification.

Table 9: BLE GATT services and characteristics implemented by our device emulator. Standard UUIDs are shown in 16-bit form on the Bluetooth SIG base UUID (`0000xxxx-0000-1000-8000-00805F9B34FB`).

Service	Characteristic	UUID	Format
Device Information (`0x180A`)	System ID	`0x2A23`	Hex
	Model Number	`0x2A24`	String
	Serial Number	`0x2A25`	String
	Firmware Revision	`0x2A26`	String
	Hardware Revision	`0x2A27`	String
	Software Revision	`0x2A28`	String
	Manufacturer Name	`0x2A29`	String
Battery (`0x180F`)	Battery Level	`0x2A19`	Percentage
Cycling Power (`0x1818`)	Cycling Power Measurement	`0x2A63`	Complex
	Cycling Power Feature	`0x2A65`	Bitmask
	Sensor Location	`0x2A5D`	Enumerated
	Wahoo control (proprietary)	`A026E005-...`	Command
Fitness Machine (`0x1826`)	Fitness Machine Feature	`0x2ACC`	Bitmask
	Indoor Bike Data	`0x2AD2`	Complex
	Supported Resistance Level Range	`0x2AD6`	Range
	Supported Power Range	`0x2AD8`	Range
	Fitness Machine Control Point	`0x2AD9`	Command
	Fitness Machine Status	`0x2ADA`	Status
Heart Rate (`0x180D`)	Heart Rate Measurement	`0x2A37`	Complex
	Body Sensor Location	`0x2A38`	Enumerated
	Heart Rate Control Point	`0x2A39`	Command

C Supplementary Results

Table 10 lists the applications referenced in §7 and §8. Tables 11 and 12 detail the pixel and tag-manager services observed in embedded WebViews (§7.2) and the apps using six or more.

Table 10: Applications referenced in the privacy analysis.

App	Package Name	Installs
Lose It!	com.fitnow.loseit	10M+
Water Tracker	com.wachanga.watertrackerpro	500K+
Pregnancy Tracker	ru.mobiledimension.kbr	10M+
Femia	femia.menstruationtracker.fertilityapp	500K+
Eka Care	eka.careeka.care	10M+
Noom	com.wsl.noom	10M+
Gymondo	de.gymondo.app.gymondo	1M+
Fitify	com.fitifyworkouts.bodyweight.workoutapp	10M+
Corporate Fitness	app.corporatefitness	10K+
MapMyRide	com.mapmyride.android2	5M+
ASICS RunKeeper	com.fitnesskeeper.runkeeper.pro	10M+
Women's Fasting	com.womenfast.intermittent.fasting.for.women	10K+
CareClinic	com.careclinicsoftware.careclinic	100K+
Calm	com.calm.android	50M+
Keto Cycle	com.kilogroup.ketocycle	500K+
Perfect Body	com.kilogroup.perfectbody	100K+
LetsGetChecked	com.letsgetchecked.app	100K+
EatMyRide	com.eatmyride.androidapp	10K+
Nutritionix Track	com.nutritionix.nixtrack	1M+
YAZIO	com.yazio.android	50M+
Oura	com.ouraring.oura	1M+
Bowflex	com.nautilus.bowflex	100K+
RunMotion Coach	com.runmotion.android	500K+
Bearable	com.bearable	500K+
FitnessBank	com.affinitybank	5K+
HeiaHeia	com.heiaheia	100K+
HiiKER	com.waymarkedtrails.hiiker	100K+
Mindbody	com.mindbodyonline.connect	5M+
MyFitnessCoach	com.myfitnesscoach	1M+
RunDay	com.hanbit.rundayfree	1M+
Cadence	com.sevenbold.cadence	50K+
Trenara	com.trenara.trenara	100K+
Trojan Health	com.volkano.trojanhealth	10K+
myCols	app.mycols	10K+

Table 11: Pixel and tag-manager services observed in our dataset, sorted by apps contacted. *Cont.* = apps that loaded the service; *ID* = apps that transmitted a real identifier to it; *Hlth* = apps that transmitted health data to it; *Both* = apps that transmitted both in a single session.

Service	Type	Cont.	ID	Hlth	Both
Meta Pixel	pixel	755	333	29	21
Google Tag Manager	tag manager	193	0	0	0
TikTok Pixel	pixel	76	25	4	1
Reddit Pixel	pixel	63	0	0	0
Segment (analytics.js)	tag manager	58	0	0	0
Snap Pixel	pixel	21	0	3	0
Microsoft Advertising UET	pixel	16	1	1	0
Criteo OneTag	pixel	15	3	0	0
LinkedIn Insight Tag	pixel	14	1	3	1
Tealium iQ	tag manager	5	0	0	0
Pinterest Tag	pixel	5	1	1	1
Twitter Universal Website Tag	pixel	4	0	1	0
Quora Pixel	pixel	3	0	0	0
Spotify Pixel	pixel	3	0	0	0
Outbrain Pixel	pixel	3	1	0	0
Taboola Pixel	pixel	2	0	1	0
Adobe Launch (DTM)	tag manager	2	0	0	0

C.1 UX Inspection Methodology

From the set of all third-party apps with associated screen recordings, we noted apps that connect to at least four integration platforms (N=69 apps). We retain only one Technogym app and remove all others (N=50) to avoid duplicate analysis of identical authorization UIs, then inspect videos for all remaining apps. After removing apps with insufficient or corrupted screen recordings (N=10), we were able to inspect 11 total third-party apps, and review integration platform authorization interactions for an average of 3 integration platforms per health app. We took field memos on designs of interest (that is, designs that potentially interfere with user autonomy

Table 12: Apps contacting six or more pixel and tag-manager services in a single session. *Tags* = distinct services contacted; *IDs* = services that received a real identifier; *Hlth* = services that received health data.

App	Tags	IDs	Hlth
com.netpulse.mobile.gymnation	9	0	1
com.wsl.noom	8	3	5
de.gymondo.app.gymondo	8	1	2
segterra.itracker.purple_android	7	0	0
com.trainerize.Trainerize	7	0	1
com.kudos.fit	6	0	0
com.gympass	6	0	0

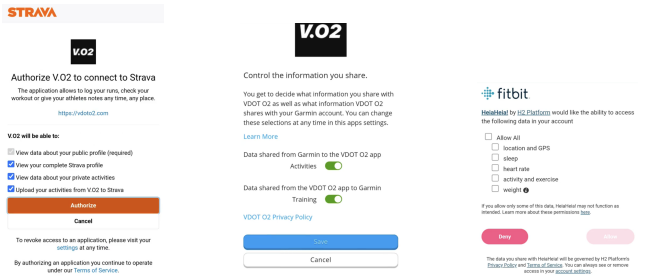

Figure 9: From left to right: Strava's OAuth flow via the RunMotion app, Garmin's OAuth flow via the RunMotion app, and Fitbit's OAuth flow via the Heiaheia app. Fitbit commits somewhat to user autonomy in choice architecture (no preselection, and provision of a bulk toggle), while Garmin and Strava offer only pre-selected, granular items requiring individual management to de-select.

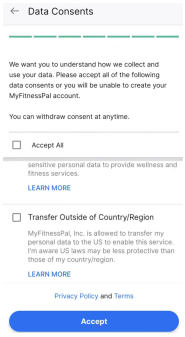

Figure 10: MyFitnessPal blocks account creation unless the user accepts US data transfer; the surrounding consent UI is otherwise granular and opt-in, so the issue is the forced permission rather than the choice architecture.

towards greater data sharing), then reviewed these designs to map them according to the Gray *et al.* ontology [33] for identifiability with prior work. A deeper investigation was not within scope for this study, as we primarily sought to contextualize the main results and consider future research opportunities.