Information Leakage in Wearable Applications

Babatunde Olabenjo(✉) and Dwight Makaroff

University of Saskatchewan, Saskatoon, SK S7N 5C9, Canada
b.olabenjo@usask.ca, makaroff@cs.usask.ca

Abstract. Wearable apps, specifically smartwatch apps, require permissions to access sensors, user profiles, and the Internet. These permissions, although not crucial for many mobile apps, are essential for health and fitness apps, as well as other wearable apps to work efficiently. Access to data on wearable devices enables malicious apps to extract personal user information. Moreover, benevolent apps can be utilized by attackers if they send private information insecurely. Many studies have examined privacy issues in smartphone apps, and very little has been done to identify and evaluate these issues in wearable smartwatch apps. Since wearable apps can reside either on the phone and watch or both, with all devices capable of accessing the Internet directly, a different dimension to information leakage is presented due to diverse ways in which these devices collect, store and transmit data.

This study classifies and analyzes information leakage in wearable smartwatch apps and examines the exposure of personal information using both static and dynamic approaches. Based on data collected from thousands of wearable applications, we show that standalone wearable apps leak less information compared to companion apps; the majority of data leaks exist in tracking services such as analytics and ad network libraries.

Keywords: Privacy · Smartwatches · Information leakage · Tracking · Wearable apps · Android

1 Introduction

Wearable devices enable unobtrusive real-time sensing in a comfortable and portable way that has influenced the medical field and other fields, such as recreation and navigation over the past decade. Researchers have proposed numerous ways that wearable devices could be used by patients and health care providers to address various healthcare challenges [1]. One of the driving forces of wearable device adoption is the smartwatch, due to its popularity and applications extending the features of mobile phone devices such as receiving push notifications on the watch, issuing voice controls, and monitoring vital signs conveniently.

Android WearOS and Apple WatchOS apps are made up of *(1)* standalone apps for the smartwatch only, *(2)* companion apps that run on the mobile phone and *(3)* embedded apps that run on the smartwatch [9]. Third-party smartwatch

G. Wang et al. (Eds.): SpaCCS 2019, LNCS 11611, pp. 211–224, 2019.
https://doi.org/10.1007/978-3-030-24907-6_17

device manufacturers, such as Fitbit, provide companion apps on mobile phones to pair with the smartwatch, facilitating data collection via multiple sources. Data collected from these devices such as heart rate, temperature, or location in real-time allows users to view their overall health information easily [4].

Privacy concerns arise when a wide range of sensitive data is released without the user's knowledge or consent [17]. Many users are aware that some of their information might be used to provide better services or collected by third-parties, but many are unaware of the type or precision of such data. For example, some users might allow their current location to be collected, but they do not know the details of the specific items released, such as GPS location, city, home address, and most visited places. As the amount of data handled by medical wearable devices grows, there is a higher risk of sensitive data exposure either during data transmission to the cloud or while data is stored locally on the device [13].

Paul and Irvine [12] investigated and compared privacy policies of wearable device services in order to understand the extent to which these services protect user privacy. Many services were found to not follow existing legislation regarding the privacy of health data. In this study, we investigate and classify information leakage in wearable smartwatch apps. We make the following contributions:

- collection and analysis of 4,017 free apps for Wear OS on Google Play store (1,894 companion and embedded, 229 standalone) and providing detailed descriptive statistics on information leakage.
- **static** analysis on each app's Dalvik bytecode to identify potential unsafe behaviours that may leak data, and **dynamic** analysis to identify information leakage via network activity tracing between wearable devices, their companion apps and the Internet.
- a comparison of information leakage in standalone, companion and embedded wearable apps based on nine potential malicious activity categories.

This paper is organized as follows. Section 2 provides a summary of related work on privacy and security in mobile and wearable apps. In Sect. 3, we give details on the dataset collection and experimental work. Section 4 provides a detailed analysis of the study done. Finally, we discuss our empirical results and draw conclusions in Sects. 5 and 6.

2 Related Work

Previous studies discussed several issues with privacy in wearable apps. Lee *et al.* [7] explored various vulnerabilities in wearable services and identified multiple security vulnerabilities in smart bands that enabled them to extract private information using three main attack scenarios (illegal device pairing, fake wearable gateway, and insecure code-based attack). Commercial smart bands expose personal and health information from both the device and the server due to wearable device misconfiguration, and if device pairing is enabled without authentication, attackers can use this to obtain user's health data.

Chauhan *et al.* [2] tested several wearable apps to uncover information leakage. They focused on traffic inspection and revealed that unique identifiers, location, credentials, and health data are transferred to the Internet. Between 4%

and 11% of apps send smartwatch-specific user activities to third-party trackers, with Google Analytics as the top tracking platform. This is due to the lack of specific permissions requirements for tracking libraries with regards to the kind of data that should be sent. Their study primarily focuses on dynamic analysis, without identifying application categories and the distribution of these leaky applications; however, results show that the majority of leakage comes from third-party advertisement and tracking libraries. Moonsamy *et al.* [10] confirms this by examining the effect of third-party ad libraries on personal information leakage in Android mobile apps, showing most personal information is sent via ad libraries. Their results show that 10% of all apps that leaked information in their study had a third-party ad library embedded in them.

Mujahid *et al.* [11] also studied wearable app permission problems in Android Wear apps and reviewed the effect on the functionality of the app. Many wearable apps have a permission mismatch problem that allows malicious apps to request permission to access personal information not required for app functionality. In other cases, a permission mismatch between the wearable device and the companion mobile app can lead to the disclosure of personal information via the mobile device without the user's proper consent on the smartwatch.

This study improves on the related work by providing a more detailed analysis of the type and categories of leaky activities by identifying 47 potential leaky activities in static and dynamic analysis. Also, we provide further insight into the distribution, category and popularity of apps with potential leaky activities in standalone, companion and embedded apps.

3 Dataset Overview

3.1 Data Collection

Figure 1 illustrates the data collection process for this study. Using a data scraper written in Python, we extracted 4,980 WearOS[1] apps from the Goko store [6] and 4,809 apps from AndroidWearCenter.[2] We then removed duplicate apps for a total of 5,599 apps. The scraper collects various app characteristics, such as package name and Google Play URL to the downloadable app which allowed us to download the APK files directly from the Google Play Store using GPlayCli.[3]

In order to have access to download APK files, 2,590 free apps were selected. Approximately 1000 apps were last updated in 2015, 2017, and 2018, respectively. Only 650 apps were last updated in 2016, and very few were older than 2015. The year of update is important as it shows recent development/maintenance. We removed 457 apps that do not have a wearable component or run only on the mobile phone, leaving a total of 2,133 apps: 1,903 companion/wearable-embedded and 230 standalone apps.

[1] Wear OS: The essential guide: https://www.wareable.com/android-wear/what-is-android-wear-comprehensive-guide (Accessed: 2019-01-29).

[2] Android Wear Center — Apps, Games, News & Watchfaces for Android Wear. http://www.androidwearcenter.com/ Accessed 2019-01-29.

[3] Matlink: https://github.com/matlink/gplaycli. (Accessed: 2019-02-03).

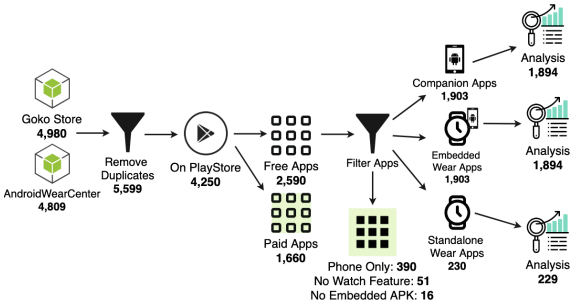


Fig. 1. Data collection and APK extraction

3.2 Extracting APKs

The publishing model for WearOS apps is by either providing a standalone APK or embedding a wearable APK inside a handheld APK.[4] In order to identify a standalone wearable app, first we unpack the APK and decode the resources using APKTool [14]. Next, we decode the manifest file and check for `hardware.type.watch` feature and `wearable.standalone` fields used in Wear 2.x to identify standalone wearable apps.

To identify embedded APKs, we checked the Manifest file to get the resource value. Next, we identify the path to the wearable APK from the XML file and then extract the wearable APK file. In some cases, identifying the paths was challenging; in such cases the `MANIFEST.MF` file in the `META-INF` folder available in all Java apps gives more details.[5]

Using regular expressions, we search through the manifest file to identify tags such as **.apk, wear, wearable* to extract the path to an embedded APK and match the package ID with the handheld APK package ID. Furthermore, the Manifest file in the extracted APK was examined for features indicating watch capability using APKTool. Of the 1,903 embedded wearable APKs extracted from their companion APK, we removed 16 handheld apps that had no embedded APKs, 51 APKs that had no `watch` feature, and 390 APKs that were designed for mobile phones only, but used the Android Wear notifications feature [16].

4 Analysis

To understand how personal information can be leaked we performed Static and Dynamic analysis. Static analysis reveals *potential* leaky activities directly from the source code without executing the app, while dynamic analysis executes the

[4] Package and distribute Wear apps: https://developer.android.com/training/wearables/apps/packaging (Accessed: 2019-01-12).

[5] https://docs.oracle.com/javase/tutorial/deployment/jar/defman.html (Accessed: 2019-01-12).

apps and observes *actual* information leakage. We explicitly separated apps into three main categories consistent with previous studies [9]:

1. *Companion/Handheld Apps* that operate on a mobile device; they usually do, however, require access to a smartwatch app to operate efficiently.
2. *Embedded/Dependent Apps* that cannot function properly on smartwatches without companion apps on mobile phones. Internet access, specific watch functions or sending data using the mobile phone.
3. *Standalone Apps* that operate independently and access the Internet directly.

We were able to perform static analysis on 1,894 of the 1,903 companion and embedded apps and all 229 standalone apps using APKTool. Figure 2 shows our analysis process. We used the *smali/baksmali* code files obtained by disassembling the Dalvik executables (.dex files). Potential leaky activities were analyzed by identifying specific tags in their *smali* code file [5]. Dynamic analysis used automated testing and collected network traffic data using Pymiproxy,[6] a lightweight Python micro interceptor proxy.

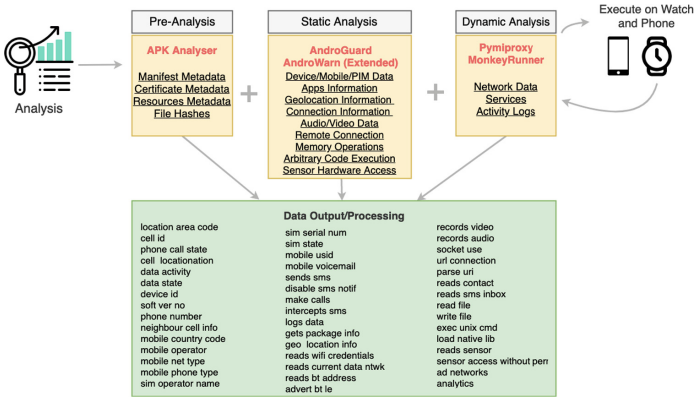


Fig. 2. Analysis process

4.1 Static Analysis

Categorizing Potential Leaky Activities. We used AndroWarn[7] as the basis for identifying activities that apps could perform which can lead to information leakage. This data was extracted from their *smali* source code to identify Android class methods used to collect location information from various providers [8]. These activities have been classified into nine different categories as follows:

[6] pymiproxy - Python Micro Interceptor Proxy - https://github.com/allfro/pymiproxy.

[7] AndroWarn: https://github.com/maaaaz/androwarn (Accessed: 2018-11-12).

- *Device/Mobile/PIM Data:* Activities that collect telephone information such as unique device IDs, phone numbers or serial numbers. Other activities include making calls, sending SMS messages and reading contacts or SMS.
- *App Information:* Log data can be accessed by background processes allowing malicious apps on older Android versions to extract personal information [15]. Also, we identify attempts to collect information about other apps on the device.
- *Geolocation Information:* Data from WiFi or GPS.
- *Connection Information:* Connection information activities such as WiFi credentials, Bluetooth addresses and the current state of the network.
- *Audio/Video Data:* Activities that use the microphone and/or camera.
- *Remote Connection:* Remote access via URI parsing and URL connection.
- *Memory Operations:* Activities that include reading/writing of files.
- *Arbitrary Code Execution:* Executing system commands, particularly on rooted devices such as loading of native libraries and execution of UNIX-like commands.
- *Sensor Hardware Access:* Several wearable apps capture sensor data. Activities in this category include registering sensor access and reading sensor data without declaring proper permission.

Companion/Handheld Apps. Companion apps have the largest number of potential leaky activities. About 93% (1,762) of apps log data, and most leaks are of the most popular types such as getting package information, reading files, opening URL connections, and listening for sensor events. The top four leaky activities make up 44.7% of the total leaks found in companion apps. Figure 3 shows the distribution of leaky activities in descending order of frequency, with a cumulative line as a percentage of the total number of apps.

Approximately 85% (1,620) of apps either read the current network connections or generate HTTP requests to an external service, while 19% (376) make socket connections to remote services. This is more common than with embedded apps because almost all embedded apps rely on the mobile phone to send watch activity to remote servers. Surprisingly, just over 11% (208) of apps get unique identifications from the device, and about 4% (84) get location information. The low percentage of apps collecting location information may be since most apps are personalization. Further analysis shows that no apps read WiFi credentials or intercept SMS messages. However, 15 apps send SMS messages, 43 apps access the mobile phone number, and 67 apps read the user's phone contacts. These are significant numbers compared to those for standalone apps.

Embedded/Standalone Applications. About 19% (364) of embedded apps still had some form of remote connection access. Although 13% (254) of embedded apps require sensor access, this is surprisingly lower than companion apps (19% (369)). This difference can be as a result of several apps using the mobile phone's sensor hardware as a primary sensor device and the watch sensor hardware as a fallback device. Also, the most popular sensor hardware is the

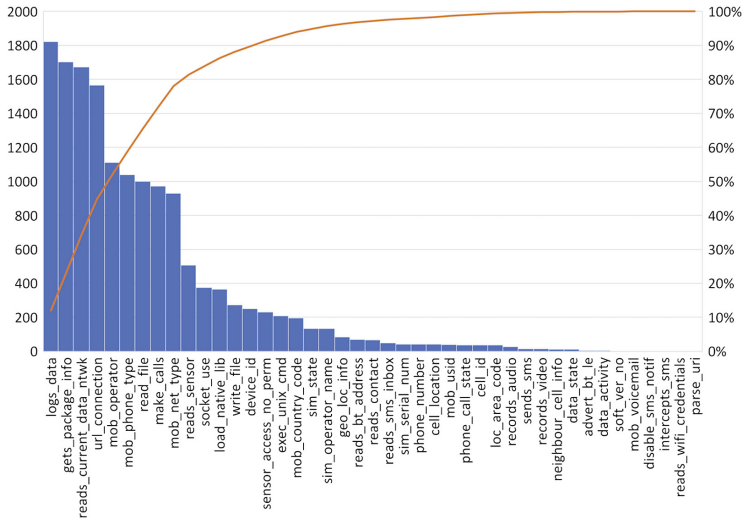


Fig. 3. Companion app leak distribution

accelerometer, available on both devices, but accessed mostly on the phone. Further analysis shows that very few embedded apps (1%) access geolocation information compared to companion apps (5%) and standalone apps (2%).

The top hardware feature used in standalone apps is the GPS with 29 apps, while just one app used the step counter feature leading to very few other sensor activities. Embedded apps, on the other hand, had the accelerometer as the top used feature (13 apps), including the step counter and heart rate features (6 and 2 apps, respectively).

Remote connections are made by 29% of standalone apps, which is much lower than companion apps (55%). This large gap is due to the limited resources available on smartwatches, because of higher bandwidth connection options available on mobile phones. A limited number of standalone apps require Internet access to operate (55%). Furthermore, 3% of apps perform device/mobile/PIM data leaks, and 16% of apps request sensor access. Table 1 highlights leakage activities from analyzed data on embedded/standalone apps.

Trackers. Using 140 tracker signatures collected from Exodus,[8] we identified over 80 different trackers from all extracted classes in each app. Google Analytics[9] was the most popular tracker used (59%). Other tracking activities were also discovered, such as ad networks, of which GoogleAds[10] was the most popular (48%). Table 2 shows the percentage of trackers identified in the three app categories in both static and dynamic analysis. An average of 3 trackers were

[8] Trackers: https://reports.exodus-privacy.eu.org/trackers/ (Accessed: 2018-11-20).
[9] Google Analytics: https://analytics.google.com/ (Accessed: 2018-12-10).
[10] Google Ads: https://www.google.com/admob/ (Accessed: 2018-12-10).

Table 1. Embedded and standalone wear app analysis

Analysis	Embedded apps (events)	PCT	Standalone apps (events)	PCT
Device/Mobile/PIM data leaks	733	2	147	3
Apps information and logs	3,645	96	424	93
Geolocation information leakage	17	1	5	2
Connection information leakage	1,495	20	181	20
Audio/video leaks	1	0	0	0
Remote connection establishment	1,090	19	199	29
Memory operations	707	19	169	37
Arbitrary code execution	196	5	15	3
Sensor hardware access	507	13	74	16

identified per app for companion apps, with some apps having as many as 22 trackers. Embedded apps however, had an average of 0.5 trackers, with 34 apps having 3 or more trackers. Standalone apps had similar averages as embedded apps, with a maximum number of 4 trackers per app, compared to embedded apps and companion apps with maximums of 8 and 22, respectively.

This data gives insight into the practice of embedding multiple ad network libraries and trackers to target ad marketing precisely. Several ad libraries use the same permissions, which prevents users from knowing exactly how many ad networks request unique permissions in the app.

Privacy Leak Distribution. To identify the distribution of apps with potential leaky activities on Google Play Store, we selected apps with Internet permission and grouped leaky apps into three main categories based on the distribution of leaky activities with a mean of 8.1, 4.4 and 5.3 and a standard deviation of 4.3, 2.4, and 2.6 for 1,894 companion, 1,894 dependent and 229 standalone apps respectively. The lower bound selected for moderate leaky activity was the average of all the means in companion, standalone and dependent apps, while the upper bound was the ceiling of the highest mean.

- **Low (L):** fewer than six leaky activities
- **Moderate (M):** between six and nine leaky activities
- **High (H):** over ten leaky activities.

Using the data collected from static analysis, we identified the categories and popularity of companion/dependent/standalone apps with leaky activities, shown in Fig. 4, disregarding 141 apps without Internet connectivity. The grey scale shows the concentration of apps in each category (companion, dependent and standalone) with darker shades representing higher numbers. Companion apps had 48% of apps with high leaky activities compared to dependent apps with 14.6% and standalone apps with 9.3%.

Category	Companion			Dependent			Standalone		
	L	M	H	L	M	H	L	M	H
ART_AND_DESIGN	0	1	1	1	0	0	0	0	0
AUTO_AND_VEHICLES	0	0	3	1	1	1	0	0	0
BOOKS_AND_REFERENCE	4	4	5	4	0	0	0	0	0
BUSINESS	0	6	6	2	1	0	0	0	0
COMMUNICATION	4	6	23	4	4	3	1	1	1
EDUCATION	2	5	10	3	1	0	0	0	0
ENTERTAINMENT	4	15	16	6	2	0	0	0	0
FINANCE	4	10	35	5	8	4	0	0	0
FOOD_AND_DRINK	0	0	2	1	0	0	0	0	0
GAMES	20	15	22	4	7	20	1	0	0
HEALTH_AND_FITNESS	2	19	39	16	9	4	0	6	0
HOUSE_AND_HOME	0	2	2	2	1	0	0	0	0
LIBRARIES_AND_DEMO	0	2	0	0	0	0	0	0	0
LIFESTYLE	12	43	20	12	4	2	2	0	0
MAPS_AND_NAVIGATION	4	13	20	10	1	0	2	0	0
MEDICAL	0	4	4	0	1	0	0	0	0
MUSIC_AND_AUDIO	1	12	24	5	2	2	2	0	1
NEWS_AND_MAGAZINES	0	6	19	13	0	1	0	0	0
PARENTING	0	1	0	0	0	0	0	0	0
PERSONALIZATION	48	171	145	60	25	2	17	60	8
PHOTOGRAPHY	2	3	4	2	2	0	0	0	0
PRODUCTIVITY	13	39	31	9	5	2	3	3	0
SHOPPING	0	3	9	1	0	0	0	1	0
SOCIAL	3	3	12	2	1	2	0	0	0
SPORTS	3	8	29	7	7	2	0	0	0
TOOLS	17	70	64	24	7	2	1	4	2
TRAVEL_AND_LOCAL	4	16	23	11	7	5	2	2	0
VIDEO_PLAYERS	1	3	6	1	1	0	1	1	0
WEATHER	4	4	14	4	1	1	4	2	0
TOTAL	152	484	588	210	98	53	36	80	12
OVERALL			1,224			361			128

Key: L=Low, M=Moderate, H=High

L=0-5 Leaky activities, **M**=6-9 Leaky activities, **H**=10+ Leaky activities

(a) Category Distribution

Downloads	Companion			Dependent			Standalone		
	L	M	H	L	M	H	L	M	H
10+	0	3	1	0	1	1	0	0	0
50+	1	2	1	0	1	1	0	0	0
100+	18	65	34	15	5	2	0	1	0
500+	21	62	36	21	5	2	2	1	0
1K+	34	102	76	32	17	10	6	7	1
5K+	14	35	43	16	9	4	3	7	1
10K+	28	93	100	36	22	7	7	31	2
50K+	16	31	38	16	5	9	6	5	2
100K+	15	56	81	31	15	3	7	17	2
500K+	2	11	31	8	5	2	1	2	1
1M+	3	16	78	16	5	6	1	6	0
5M+	0	3	22	10	2	3	1	0	0
10M+	0	4	36	8	5	2	1	3	0
50M+	0	0	3	1	0	0	0	0	0
100M+	0	1	6	0	1	1	0	0	1
500M+	0	0	1	0	0	0	0	0	0
1B+	0	0	1	0	0	0	1	0	2
TOTAL	152	484	588	210	98	53	36	80	12
OVERALL			1,224			361			128

Key: L=Low, M=Moderate, H=High

L=0-5 Leaky activities, **M**=6-9 Leaky activities, **H**=10+ Leaky activities

(b) Popularity Distribution

Fig. 4. Category and popularity distribution of leaky apps

Personalization had more apps overall, while *Communication*, *Finance*, *Health & Fitness*, *Music & Audio*, *Travel & Local* and *Weather* had the largest proportion of apps with a high number of leaky activities per category (Fig. 4a). Dependent apps had fewer categories containing apps with a high number of leaky activities with *Arcade Games* as the top category. Interestingly, dependent apps had a low number of leaky activities compared to companion apps as dependent apps rely on their companion apps to compute, aggregate and transmit data. Standalone apps, however, had few apps with a high number of leaky activities per category, but more apps with a moderate number of leaky activities (75% in the personalization category).

Quantifying the popularity of vulnerable apps presents an insight into the issue of privacy leakage. In Fig. 4b, we identified the popularity of the low, moderate and high leaky activities in apps based on the number of downloads. Our results show that there were several moderate leaky activities in companion apps with low popularity; moderate to high popularity apps had higher ratios of apps with a high number of leaky activities. In addition, two apps with a high number of leaky activities were very popular. Just because there are leaky activities in apps doesn't mean the app is problematic. These activities must be *used* in the operation of the app, and further analysis is needed to quantify the vulnerability. The categorization of leaky activities here gives an insight into what categories of apps have potentially more leaky activities than others.

Dependent apps had a higher percentage of apps with low leaky activities and reasonably high popularity. Although very few dependent apps have low leaky activities, they require a companion app installation which increases the risk of information leakage. Among apps with more than 50 million downloads, 15/18 (83%) had a high number of leaky activities.

4.2 Dynamic Analysis

Inspecting network traffic generated by the apps allows us to check if personal information is being sent to an external source. We executed each companion app along with their embedded versions simultaneously on a mobile device, and an Android wear device for 15 s with 10 random touch interactions to simulate a more realistic action and allow time for network requests to be made during the touch events.

The handheld/companion app was executed on a Google Pixel XL with 2 dual-core Kyro processors, 128 GB storage capacity, 4 GB RAM running Android 8.0 while the wearable app was executed on a paired Motorola Moto 360 Sport with Quad-core 1.2 GHz Cortex-A7, 4 GB, 512 MB RAM running WearOS 2.0. Standalone apps were executed independently on the smartwatch and network packets were captured including HTTPS traffic.

A Python script was written with the use of MonkeyRunner[11] to execute both apps on the mobile phone and smartwatch devices while Pymiproxy was used to intercept and extract network traffic for inspection. We executed 463 apps that had 10 or more potentially leaky activities discovered from static analysis with `android.permission.INTERNET`[12] set in their Manifest file.

Our tests showed that 73% of companion apps selected made substantial Internet requests (five or more GET/POST requests including socket connections) while 8% of embedded apps sent requests to remote servers. This low number of embedded apps connecting to remote servers is not surprising because embedded apps depend on companion apps to send traffic to remote servers [2]. We further evaluated 128 out of the 229 standalone apps with Internet access permission and identified 13% made connections to remote servers.

Trackers. Network traffic generated by the apps were inspected and over 30 trackers were identified. Several companion apps sent data to more than one tracker with Google CrashLytics, Google DoubleClick and Google Analytics as the top trackers actually used. 43%, 33% and 27% of apps sent requests to Google CrashLytics, Google DoubleClick, and Google Analytics respectively. In addition, 22% of apps sent data to Facebook Login, Facebook Analytics, and Facebook Ads.

[11] Google Developer Support - Monkeyrunner: https://developer.android.com/studio/test/monkeyrunner/ (Accessed: 2019-02-20).

[12] Google Developer Support - Connect to the network: https://developer.android.com/training/basics/network-ops/connecting (Accessed: 2018-11-21).

Table 2. Top trackers, analytics and Ad networks

Static Analysis									Dynamic Analysis								
Companion/Handheld			Dependent/Embeded			Standalone			Companion/Handheld			Dependent/Embeded			Standalone		
Trackers	**Apps**	**%**	**Trackers**	**Apps**	**%**	**Trackers**	**Apps**	**%**	**Trackers**	**Apps**	**%**	**Trackers**	**Apps**	**%**	**Trackers**	**Apps**	**%**
Google Analytics	1116	59	*Google Analytics*	489	26	*Firebase Analytics*	53	23	*Google CrashLytics*	147	43	*Google CrashLytics*	16	42	*Google Analytics*	8	47
Google Ads	904	48	*Google Ads*	68	4	*Google CrashLytics*	16	7	*Google DoubleClick*	112	33	*Google Ads*	14	37	*Google CrashLytics*	6	35
Google DoubleClick	834	44	*Firebase Analytics*	68	4	*Google Analytics*	15	7	*Google Analytics*	93	27	*Google Analytics*	5	13	*Google Ads*	2	12
Firebase Analytics	735	39	*Google DoubleClick*	67	4	*Google Ads*	8	3	*FB Login*	74	22	*Flurry*	2	5	*FB Analytics*	1	6
Google CrashLytics	262	14	*Google CrashLytics*	31	2	*Google DoubleClick*	8	3	*FB Analytics*	74	22	*Unity3d Ads*	1	3	*FB Audience*	1	6
Flurry	168	9	*Flurry*	22	1	*FB Analytics*	1	0.4	*FB Ads*	74	22				*FB Notifications*	1	6
FB Login	144	8	*ChartBoost*	19	1	*AppsFlyer*	1	0.4	*FB Share*	74	22				*FB Login*	1	6
FB Share	137	7	*FB Analytics*	3	0.2	*Flurry*	1	0.4	*FB Places*	74	22				*FB Ads*	1	6
FB Analytics	132	7	*FB Share*	3	0.2				*FB Audience*	74	22				*FB Places*	1	6
Inmobi	108	6	*FB Login*	2	0.1				*FB Notifications*	74	22				*FB Share*	1	6
FB Ads	93	5	*FB Places*	2	0.1				*Google Ads*	27	8						
Twitter MoPub	77	4	*Demdex*	2	0.1				*Flurry*	25	7						
FB Places	55	3	*ComScore*	2	0.1				*Unity3d Ads*	24	7						
Amazon Advertisement	49	3	*AppNexus*	1	0.1				*Yandex Ad*	11	3						
ChartBoost	42	2	*Urbanairship*	1	0.1				*AppMetrica*	11	3						
Unity3d Ads	40	2	*Omniture*	1	0.1				*Urbanairship*	10	3						
Moat	30	2	*AppMetrica*	1	0.1				*MixPanel*	9	3						
Millennial Media	29	2	*New Relic*	1	0.1				*Nexage*	9	3						
Adjust	23	1	*Google Tag Manager*	1	0.1				*Adjust*	9	3						
HockeyApp	22	1							*Krux*	8	2						

Although few embedded apps made remote connections, Google Analytics and Google Ads were top trackers, similar to their companion apps. Standalone apps also had a similar percentage of trackers as the companion and embedded apps. Table 2 provides a full breakdown of trackers, analytics and ad networks identified in network traffic generated during dynamic analysis.

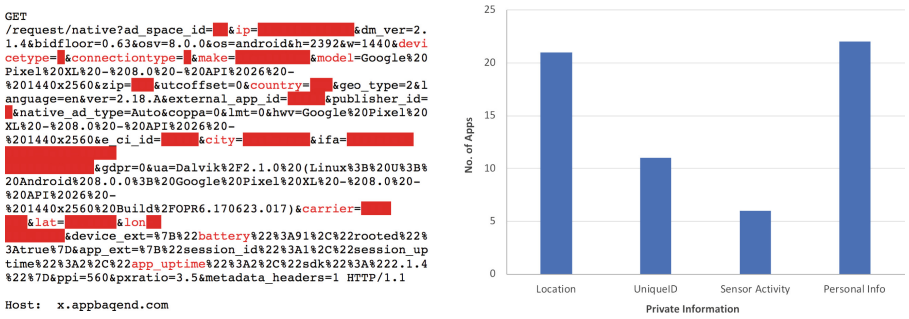
Companion apps had 2 trackers per app, on average, while embedded and standalone apps had an average of 1, and 1.4. 21 companion apps had 10 or more trackers while 1 embedded (`com.sofascore.results`) and 1 standalone app (`ch.pete.wakeupwell`) had 2 and 8 trackers, respectively.

Personal Information Leakage. By inspecting the app traffic generated, we classified personal information into four main groups as described below:

- *Location:* country code, IP address, GPS location data, e.g. GPS coordinates, city, state, and network location data.
- *UniqueID:* Device unique IDs identified includes MAC addresses, device IDs, IMEI codes and device hashes.
- *Sensor Activity:* step-count, device orientation, heart rate and sensor data.
- *Personal Information:* User's personal information such as health data, address, marital status, age, sex, date of birth, gender, email, income.

Figure 5a shows a sample data record captured from a single companion app sending personal location information as well as app/battery usage. This data is

mostly used by advertisers to improve ad targeting. However, very few users are aware of the amount of data collected by advertisers [3]. Although few standalone apps leaked personal information, 2/17 standalone apps that generated network traffic tracked device location, and just one app collected unique IDs.



(a) Personal Information Leakage Example (b) Observed Information Leakage

Fig. 5. Companion app personal information leaks

Embedded apps did not, however, send any personal information to the Internet because almost all data collected by embedded apps were forwarded to their companion apps. Companion apps sent the majority of personal information as shown in Fig. 5b. About 4.8% of companion apps sent personal information such as email, age, gender and others while 4.5% of apps sent location information. Also, 2.4% of companion apps sent unique IDs, and we identified about 1.3% of apps sending sensor activities. The majority of sensor activities observed were the accelerometer and device orientation data.

5 Discussion

By studying 4,017 (1,894 companion, 1,894 embedded, 229 standalone) apps independently in the Android smartwatch market, we observed that the majority of smartphone companion apps send more sensitive information compared to standalone or embedded apps. Static analysis revealed that 12% of companion apps, 11% of embedded apps, and about 9% of standalone apps register sensor hardware in their manifest file. Many of these apps do not even use registered embedded sensors in their code, as observed during static analysis due to third-party libraries registering sensors despite not being required by the app, enabling unauthorized access/transmission of associated data.

Third-party trackers are the major cause of personal information leakage, as shown in Table 2. Additionally, apps with more leaky activities were popular in companion apps, with over 1 million downloads, as seen in Fig. 4. We identified over 84 different trackers in companion apps with standalone apps having the

least number of trackers (8) due to the limited smartwatch developer ecosystem compared to the smartphone market. Also, many ad networks are yet to develop effective ad monetization schemes for developers on the smartwatch market.

Further classification was done based on the number of leaky activities apps contain, and we identified both the popularity of these apps and their app categories on the Play Store. Our results show that there are specific categories with more risky behaviors than others, as shown in Fig. 4, suggesting that the wearable app community should adopt further verification mechanisms in apps of such categories such as Personalization, Health & Fitness, and Tools. Although the number of leaky activities does not necessarily mean the app is potentially dangerous, the fact that an app has high leaky activities should trigger a further investigation into the app, its operation, and permissions, because such apps are likely to leak more private information in actual operation.

Unlike mobile apps, wearable apps easily collect data regularly because they are in close contact with the user, and with the help of a companion app could potentially collect vast quantities of data and store them remotely which can lead to the exposure of sensitive information without the user's knowledge. Our results show that leaky applications are currently popular, with many having over 10 million downloads. With the growing number of potential leaky applications currently existing on the app store, simple privacy-preserving actions such as disabling unnecessary app permissions, avoiding fingerprint authentication on apps and encrypting phone data, can help users potentially reduce the amount of information leaked.

6 Conclusion and Future Work

Static and dynamic analysis reveals that third-party tracking libraries account for the bulk of information leakage in all categories, while companion apps leak the most data. Due to the limited resources available in smartwatches and the limited number of third-party tracking services developed for WearOS app developers, standalone apps are less likely to send sizeable sensitive information remotely. In this study, we classify leaky activities in wearable apps and identify the popularity of leaky apps based on the number of downloads and category on the app store, allowing researchers to identify where leaky apps are most likely to exist. However, fine-grained analysis of each leak in the applications has been left for future work and is beyond the scope of this study.

Further automated/manual testing is necessary to verify the degree to which potential risk is correlated with actual damage. This challenge drives further investigation on how information leakage can be mitigated by analyzing the efficacy of tracking settings and permission models, enabling users to control how their personal information is collected. If that is a practical approach, then we can develop models to allow users to fully control access to their personal information in real-time from wearable devices.

References

1. Boillat, T., Rivas, H., Wac, K.: "Healthcare on a Wris": increasing compliance through checklists on wearables in obesity (self-)management programs. In: Rivas, H., Wac, K. (eds.) Digital Health. HI, pp. 65–81. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-61446-5_6
2. Chauhan, J., Seneviratne, S., Kaafar, M.A., Mahanti, A., Seneviratne, A.: Characterization of early smartwatch apps. In: PerCom Workshops, pp. 1–6. Sydney, Australia, March 2016
3. Chen, G., Meng, W., Copeland, J.: Revisiting mobile advertising threats with MAdLife. In: The World Wide Web Conference, WWW 2019, pp. 207–217, San Francisco, CA, May 2019
4. Fafoutis, X., Marchegiani, L., Papadopoulos, G.Z., Piechocki, R., Tryfonas, T., Oikonomou, G.: Privacy leakage of physical activity levels in wireless embedded wearable systems. IEEE Signal Process. Lett. **24**(2), 136–140 (2017)
5. Hou, S., Ye, Y., Song, Y., Abdulhayoglu, M.: HinDroid: an intelligent Android malware detection system based on structured heterogeneous information network. In: KDD 2017, Halifax, Canada, pp. 1507–1515, August 2017
6. Korner, J., Hitzges, L., Gehrke, D.: Goko Store: Home. https://goko.me/
7. Lee, M., Lee, K., Shim, J., Cho, S., Choi, J.: Security threat on wearable services: empirical study using a commercial smartband. In: ICCE-Asia, Seoul, South Korea, pp. 1–5, October 2016
8. Li, X., Dong, X., Liang, Z.: A usage-pattern perspective for privacy ranking of Android apps. In: Prakash, A., Shyamasundar, R. (eds.) ICISS 2014. LNCS, vol. 8880, pp. 245–256. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13841-1_14
9. Liu, R., Lin, F.X.: Understanding the characteristics of Android wear OS. In: ACM Mobisys, Singapore, Singapore, pp. 151–164, June 2016
10. Moonsamy, V., Batten, L.: Android applications: data leaks via advertising libraries. In: International Symposium on Information Theory and its Applications, Melbourne, Australia, pp. 314–317, October 2014
11. Mujahid, S.: Detecting wearable app permission mismatches: a case study on Android wear. In: 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, pp. 1065–1067, September 2017
12. Paul, G., Irvine, J.: Privacy implications of wearable health devices. In: SIN 2014, Glasgow, UK, pp. 117:117–117:121, September 2014
13. Sun, W., Cai, Z., Li, Y., Liu, F., Fang, S., Wang, G.: Security and privacy in the medical Internet of Things: a review. Secur. Commun. Netw. **2018**, 1–9 (2018)
14. Tumbleson, C., Winiewski, R.: Apktool - a tool for reverse engineering 3rd party, closed, binary Android apps. https://ibotpeaches.github.io/Apktool/
15. Wu, S., Zhang, Y., Jin, B., Cao, W.: Practical static analysis of detecting intent-based permission leakage in Android application. In: IEEE ICCT, Chengdu, China, pp. 1953–1957, October 2017
16. Zhang, H., Rounte, A.: Analysis and testing of notifications in Android wear applications. In: International Conference on Software Engineering, Buenos Aires, Argentina, pp. 347–357, May 2017
17. Zhang, K., Ni, J., Yang, K., Liang, X., Ren, J., Shen, X.S.: Security and privacy in smart city applications: challenges and solutions. IEEE Commun. Mag. **55**(1), 122–129 (2017)