The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors

Anupam Das
North Carolina State University
anupam.das@ncsu.edu

Gunes Acar
Princeton University
gunes@princeton.edu

Nikita Borisov
University of Illinois at Urbana-Champaign
nikita@illinois.edu

Amogh Pradeep
Northeastern University
a.pradeep@northeastern.edu

ABSTRACT

We present the first large-scale measurement of smartphone sensor API usage and stateless tracking on the mobile web. We extend the OpenWPM web privacy measurement tool to develop OpenWPM-Mobile, adding the ability to emulate plausible sensor values for different smartphone sensors such as motion, orientation, proximity and light. Using OpenWPM-Mobile we find that one or more sensor APIs are accessed on 3 695 of the top 100K websites by scripts originating from 603 distinct domains. We also detect fingerprinting attempts on mobile platforms, using techniques previously applied in the desktop setting. We find significant overlap between fingerprinting scripts and scripts accessing sensor data. For example, 63% of the scripts that access motion sensors also engage in browser fingerprinting.

To better understand the real-world uses of sensor APIs, we cluster JavaScript programs that access device sensors and then perform automated code comparison and manual analysis. We find a significant disparity between the actual and intended use cases of device sensor as drafted by W3C. While some scripts access sensor data to enhance user experience, such as orientation detection and gesture recognition, tracking and analytics are the most common use cases among the scripts we analyzed. We automated the detection of sensor data exfiltration and observed that the raw readings are frequently sent to remote servers for further analysis.

Finally, we evaluate available countermeasures against the misuse of sensor APIs. We find that popular tracking protection lists such as EasyList and Disconnect commonly fail to block most tracking scripts that misuse sensors. Studying nine popular mobile browsers we find that even privacy-focused browsers, such as Brave and Firefox Focus, fail to implement mitigations suggested by W3C, which includes limiting sensor access from insecure contexts and cross-origin iframes. We have reported these issues to the browser vendors.

CCS CONCEPTS

• **Security and privacy** → *Web application security*; *Privacy protections*;

KEYWORDS

Sensors; Mobile browser; On-line tracking; Fingerprinting

ACM Reference Format:
Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243860

1 INTRODUCTION

The dominant mode of web browsing has shifted towards mobile platforms—2016 saw mobile web usage overtake desktop [74]. Today's smartphones come equipped with a multitude of sensors including accelerometers, gyroscopes, barometers, proximity and light sensors [67]. Augmented reality, indoor navigation and immersive gaming are some of the emerging web applications possible due to the introduction of sensors. The Web's standardization body, the W3C, has thus introduced standards to define how to make sensor data accessible to web applications [80].

Access to the sensors, however, can also create new security and privacy vulnerabilities. For example, motion sensors can be exploited to infer keystrokes or PIN codes [14, 51]. Ambient light level readings can be exploited for sniffing users' browsing history and stealing data from cross-origin iframes [61]. Motion sensors have also shown to be uniquely traceable across websites, allowing stateless tracking of users [9, 19, 23]. While the W3C's sensor specifications list these and other security and privacy concerns, they do not mandate countermeasures. In practice, mobile browsers allow access to these sensors without explicit user permission, allowing surreptitious access from JavaScript.

In order to better understand the risks of sensor access, we conduct an in-depth analysis of real-world uses and misuses of the sensor APIs. In particular, we seek to answer the following questions: 1) what is the prevalence of scripts that make use of sensors? 2) what are the common use cases for accessing sensors? 3) are sensors used by third-party tracking scripts, specifically those script which engage in fingerprinting? 4) how effective are existing privacy countermeasures in thwarting the use of sensors by untrusted scripts?

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243860

To answer these questions, we perform the first large-scale measurement study of the mobile web with a focus on sensor APIs. We extend the OpenWPM [31] measurement platform to study the mobile web, adding emulation of mobile browsing behavior and browser APIs. We call this extension *OpenWPM-Mobile* and have released its source code publicly. Using the JavaScript and HTTP instrumentation data provided by OpenWPM-Mobile, we survey the Alexa top 100K sites. We measure the sensor API access patterns, in combination with stateless tracking techniques including canvas, battery, WebRTC and AudioContext fingerprinting. To understand how sensors are being used in the wild, we develop a clustering scheme to group similar scripts and then perform manual analysis to identify use cases. Furthermore, we measure how popular tracking protection lists perform against tracking scripts that make use of sensors.

Below we present a summary of our findings:

Large-scale measurement of sensor API usage (§4). We find that on 3 695 of the Alexa top 100K sites at least one of motion, orientation, proximity, or light sensor APIs is accessed. By emulating real sensor data, we were able to determine that many third-party scripts send raw sensor data to remote servers.

Study of sensor API use through clustering (§5). By clustering scripts based on features extracted from JavaScript instrumentation data, we find a significant disparity between the intended use cases of sensors (as drafted by the W3C) and real-world uses. Sensor data are commonly used for tracking and analytics, verifying ad impressions, and distinguishing real devices from bots.

Measurement of fingerprinting on the mobile web (§6.1). We present the first mobile web measurement of the various fingerprinting techniques including canvas, WebRTC, AudioContext, and battery fingerprinting. We find a significant overlap between the fingerprinting scripts and the scripts accessing sensor APIs, indicating that sensor data is used for tracking purposes. For example, we found that 63% of the scripts that access motion sensors also perform canvas fingerprinting.

Evaluation of existing countermeasures (§6.2 and §6.3). We evaluate the efficacy of existing countermeasures against tracking scripts that use sensor APIs. We measure the rate of blocking by three popular tracking protection lists: EasyList, EasyPrivacy, and Disconnect. We find that these lists block the most prevalent scripts that access sensor; however, they only block 2.5–3.3% of the scripts overall. We also study the sensor access behavior of nine popular browsers, and find that browsers, including more privacy oriented ones such as Firefox Focus and Brave, fail to follow the W3C recommendation of disallowing access from insecure origins and cross-origin iframes. We have reported these issues to the specific browser vendors.

2 BACKGROUND AND RELATED WORK

2.1 Mobile Sensor APIs

Our study focuses on the following sensor APIs for device motion, orientation, proximity, and ambient light. Other sensors commonly present on modern mobile devices such as magnetometer, barometers, and infrared sensors are left out as they are not supported by browsers. We provide a brief description of the sensors below:

Motion. (devicemotion [68]) Provides acceleration and rotation rate along three axes using MEMS accelerometers and gyroscopes. Data type is double-precision floating-point with acceleration values expressed in ms^{-2} unit and rotation rates expressed in $rads^{-1}$ or $degs^{-1}$ unit.

Orientation. (deviceorientation [80]) Provides *alpha*, *beta* and *gamma* components which correspond to orientation along the Z, X and Y axes, respectively. Data type is double-precision floating-point, specified in *deg* unit.

Proximity. (deviceproximity [72]) Detects if the phone is close to ear during a call based on light and infrared sensors. Provides double-precision floating-point readings in *cm* units.

Ambient Light. (devicelight [71]) Provides the ambient light level readings in *lux* units.

To access sensor data, a script registers an event handler by calling the addEventListener function with the specific sensor event and event handler functions as arguments. The event handler function is then called whenever new sensor data is available. A sample code snippet for registering and accessing motion sensor is given below:

```
window.addEventListener("devicemotion", motionHandler);
function motionHandler(evt){
  // Access Accelerometer Data
  ax = evt.accelerationIncludingGravity.x;
  ay = evt.accelerationIncludingGravity.y;
  az = evt.accelerationIncludingGravity.z;
  // Access Gyroscope Data
  rR = evt.rotationRate;
  if (rR != null){
    gx = rR.alpha;
    gy = rR.beta ;
    gz = rR.gamma;
  }
}
```

Note that proximity and ambient light sensors were only supported by Firefox and their support has been deprecated. Nevertheless, our study finds some usage of theses sensors across the web.

2.2 Different Uses of Sensor Data

W3C has listed the following use cases for device sensors [87]:

- Light: controlling smart home lighting, checking sufficient light level at work space, calculating camera settings (aperture, shutter speed, ISO) and light-based gesturing.
- Proximity: detecting when device is held close to the mouth or ear (e.g., WebRTC-based voice call application).
- Motion and Orientation: virtual and augmented reality (head movement tracking), immersive gaming, activity and gesture recognition, fitness monitoring, 3D scanning and indoor navigation.

The potential uses of sensor APIs are not limited to the cases listed above. In section 5.4, we will summarize the different uses of the sensor APIs found in the wild.

2.3 Related Work

Sensor Exploitation. Prior studies have shown a multitude of creative ways to exploit sensor data: inferring keystrokes and PIN codes using motion sensors [14, 51, 63, 88]; capturing and reconstructing audio signals using gyroscopes [53]; inferring whether you are walking, driving, or taking the subway using motion sensors [73, 79]; tracking the metro ride or inferring the route that was driven using motion data [39, 40]; sniffing users' browsing history and stealing data from cross-origin frames using ambient light level readings [61]; extracting a spatial fingerprint of the surroundings using a combination of acoustic and motion sensors [6]; linking users' incognito browsing sessions to their normal browsing sessions using the timing of the devicemotion event firings [82].

Browser Fingerprinting. Mayer [49] first explored the idea of using browser "quirks" to fingerprint users; the the Panopticlick project was the first to show that browser fingerprinting can be done effectively at scale [29]. In 2012, Mowery and Shacham introduced canvas fingerprinting, which uses HTML5 canvas elements and WebGL API to fingerprint the fonts and graphic rendering engine of browsers [56]. Finally, several measurement studies have shown the existence of these advanced tracking techniques in the wild [1, 2, 31, 58, 60]. Recently, browser extensions have been shown to be fingerprintable [78]. Cao et al. recently proposed ways in which it is possible to identify users across different browsers [15]. Vastel et al. have also shown that in spite of browser fingerprints evolving over time they can still be linked to enable long-term tracking [85].

As mobile browsing became more common, researchers explored different ways to fingerprint mobile devices and browsers. Hardware and software constraints on mobile platforms often lower the fingerprinting precision for mobile browsers [29, 41, 76]. In 2016, however, Laperdrix et al. showed that fingerprinting mobile devices can be effective, mainly thanks to user agent strings, and emojis, which are rendered differently across mobile devices [48]. Others have looked at uniquely identifying users by exploiting the mobile configuration settings, which are often accessible to mobile apps [46].

Researchers have also studied ways to mitigate browser fingerprinting. Privaricator [59] and FPRandom [47] are two approaches that add randomness to browser attributes to break linkability across multiple visits. Besson et al. formalized randomization defense using quantitative information flow [8]. FP-Block [81] is another countermeasure that defends against cross-domain tracking while still allowing first-party tracking to improve usability. Some browsers such as Tor browser and Brave by default protect against various fingerprinting techniques [11, 65].

Device Fingerprinting. It is also possible to use unique characteristics of the user's hardware instead of, or in addition to, browser software properties for fingerprinting purposes. One of the early and well-known results showed that computers can be uniquely fingerprinted by their clock skew rate [55]. Later on, researchers were able to show that such tracking can be done on the Internet using TCP and ICMP timestamps [44].

In recent years, researchers have looked into fingerprinting smartphones through embedded sensors. Multiple studies have looked at uniquely characterizing the microphones and speakers embedded in smartphones [9, 18, 89]. Motion sensors such as accelerometers and gyroscopes have also shown to exhibit unique properties, enabling apps and websites to uniquely track users online [9, 19, 20, 23]. The HTML5 battery status API has also been shown to be exploitable; specially old and used batteries with reduced capacities have been shown to potentially serve as tracking identifiers [62].

Taking a counter perspective, researchers have also explored the potential of using browser and device fingerprinting techniques to augment web authentication [4, 66, 83]. In this setting, fingerprints collected using the sensor or other APIs serve as an additional factor for authentication. Device fingerprinting has also been proposed as a way to distinguish users browsing real devices from bots or other emulated browsers [13].

In this paper we focus on the tracking-related use of sensors embedded in smartphones. Our goal is not to introduce new fingerprinting schemes or evaluate the efficacy of existing techniques. Rather we identify the real-world uses of sensors APIs by analyzing data from the first large-scale mobile-focused web privacy measurement. Moreover, we highlight the substantial disparity between the intended and actual use of smartphone sensors.

3 DATA COLLECTION AND METHODOLOGY

3.1 OpenWPM-Mobile

Our data collection is based on OpenWPM-Mobile, a mobile-focused measurement tool we built by modifying OpenWPM web measurement framework [31].[1] OpenWPM has been developed to measure web tracking for desktop browsers and hence it uses the desktop version of Firefox browser as a part of its platform. To capture the behavior for mobile websites, we heavily modified OpenWPM platform to imitate a mobile browser. This was essential for performing large-scale crawls of websites, as mobile browsers have more limited instrumentation capability. We specifically emulate Firefox on Android, as it uses the same Gecko layout engine as the desktop Firefox used in the crawls; it is also the only browser that supports all four of the sensor APIs that we study.[2]

We extended OpenWPM's JavaScript instrumentation to intercept access to sensor APIs. In particular, we logged calls to the addEventListener function, along with the function arguments and stack frames. We also used OpenWPM's standard instrumentation that allowed us to detect fingerprinting attempts including canvas fingerprinting, canvas-font fingerprinting, audio-context fingerprinting, battery fingerprinting and WebRTC local IP discovery [31].

Sites are known to produce different pages and scripts for mobile browsers; to ensure that we would see the mobile versions, we took several steps to realistically imitate Firefox for Android. This involved overriding navigator object's user agent, platform, appVersion and appCodeName strings; matching the screen resolution, screen dimensions, pixel depth, color depth; enabling touch status; removing plugins and supported MIME types that may indicate a desktop browser. We also adjusted the preferences used

[1] The source code for OpenWPM-Mobile can be found at: https://github.com/sensor-js/OpenWPM-mobile

[2] Firefox released a version that disables devicelight and deviceproximity events on May 9th, 2018 [43].

to configure Firefox for Android such as hiding the scroll bars and disabling popup windows. We relied on the values provided in the mobile.js[3] script found in the Firefox for Android source code repository. To mitigate detection by font-based fingerprinting [2, 31], we uninstalled all fonts present on crawler machines and installed fonts extracted from a real smartphone (Moto G5 Plus) with an up-to-date Android 7 operating system.

To make sure that our instrumented browser looked realistic, we used fingerprintjs2 [84] library and EFF's Panopticlick test suite [30] to compare OpenWPM-Mobile's fingerprint to the fingerprint of a Firefox for Android running on a real smartphone (Moto G5 Plus). We found that OpenWPM-Mobile's fingerprint matched the real browser's fingerprint except Canvas and WebGL fingerprints. Since these two fingerprints depend on the underlying graphics hardware and exhibit a high diversity even among the mobile browsers [48], we expect that sites are unlikely to disable mobile features solely based on these fingerprints.

3.2 Mimicking Sensor Events

Since the browser we used for crawling is not equipped with real sensors, we added extra logic into OpenWPM-Mobile to trigger artificial sensor events with realistic values for all four of the device APIs. We ensured that the sensor values were in a plausible range by first obtaining them from real mobile browsers through a test page. To allow us to trace the usage of these values through scripts, we used a combination of fixed values and a small random noise. For instance, for the alpha, beta and gamma components of the deviceorientation event, we used 43.1234, 32.9876, 21.6543 as the base values and added random noise with five leading zeros (e.g., 0.000005468). This fixed base values allowed us to track sensor values that are sent within the HTTP requests. The random noise, on the other hand, prevented unrealistic sensor data with fixed values.

3.3 Data Collection Setup

We crawl the Alexa top 100K ranked[4] websites [5] using OpenWPM-Mobile. The crawling machines are hosted in two different geographical locations; one in the United States, at the University of Illinois, and the other in Europe, at a data center in Frankfurt. We conducted two separate crawls of the top 100K sites in US (producing crawls US1, collected May 17–21, 2018 and US2, collected May 27–June 1, 2018) and one from Germany (EU1, collected May 17–21, 2018). US1 is our default dataset and thus majority of our analysis is evaluated on US1; the other crawls are analyzed in section 4.3. Figure 1 highlights the overall data collection and processing pipeline. We are making our data sets available to other researchers [17].

Table 1: Overview of different types of low-level features.

Feature name format	Operation
get_*symbolName*	Property lookup
set_*symbolName*	Property assignment
call_*functionName*	Function call
addEventListener_*eventName*	addEventListener call

3.4 Feature Extraction

To be able to characterize and analyze script behavior, we first represent script behavior as vectors of binary features. We extract features from the JavaScript and HTTP instrumentation data collected during the crawls. For each script we extract two types of features: low- and high-level, as described below.

Low-level features: Low-level features represent browser properties accessed and function calls made by the script. OpenWPM instruments various browser properties relevant to fingerprinting and tracking using JavaScript getter and setter methods. We define two corresponding features: get_*SymbolName* that is set to 1 when a particular property is accessed and set_*SymbolName* that is set when a property is written to. For example, a script that reads the user-agent property would have the get_window.navigator.userAgent feature, and a script that sets a cookie would have the set_window.document.cookie feature.

OpenWPM also tracks a number of calls to JavaScript APIs that are related to fingerprinting, such as HTMLCanvasElement. toDataURL and BatteryManager.valueOf. We represent calls with a call_functionName feature. We create a special set of features for the addEventListener call to capture the type of event that the scripts are listening for. For example:

window.addEventListener("devicemotion",. . .)

would result in the addEventListener_devicemotion feature being set for the script. The four types of low-level features are summarized in Table 1.

High-level features: The high-level features capture the tracking related behavior of scripts. The features include whether a

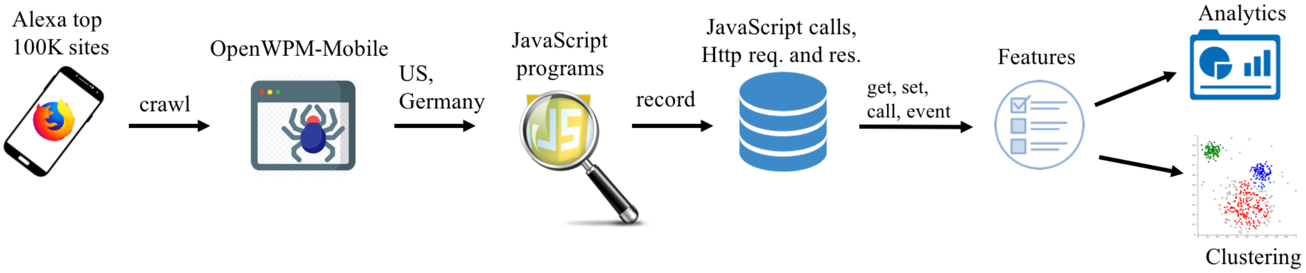

Figure 1: Overview of data collection and processing work flow.

[3] https://dxr.mozilla.org/mozilla-esr45/source/mobile/android/app/mobile.js
[4] Using rankings dated May 12, 2018.

script is using different browser fingerprinting techniques, such as canvas or audio-context fingerprinting, and whether the script is blocked by certain adblocker list or not. We use techniques from existing literature [1, 31] to detect fingerprinting techniques. We check the blocked status of the script by using three popular adblocking/tracking protection lists: EasyList [27], EasyPrivacy [28], and Disconnect [25]. The full list of high-level features are given in Table 2.

3.5 Feature Aggregation

We produce a feature vector for each script loaded by each site in the crawl. For analysis purpose, we aggregate these feature vectors in three different ways: *site, domain,* and *url*. Site-level aggregation considers the features used by all the scripts loaded by a given site. Domain-level aggregation captures all the scripts (across all sites) that are served from a given domain, to identify major players who perform sensor access. We use the Public Suffix + 1 (PS+1) domain representation, which are commonly used in the web privacy measurement literature to group domains issued to a single entity [50, 57]. We also group accesses by script URL to capture the use of the same script across different sites. When performing this grouping, we discard the fragment and query string URL components [7] (i.e., the part of the URL after the ?, & or # characters), as these are often used to pass script parameters or circumvent caching.

When performing this aggregation, we essentially compute a binary OR of the feature vectors of the individual instances that we incorporate. In other words, if any member of the grouping exhibits a certain feature, the feature is assigned to a script. For example, if any script served by a given domain performs canvas fingerprinting, we assign the canvas_fingerprinting feature for that domain.

4 MEASUREMENT RESULTS

In this section, we will first highlight the overall prominence of scripts accessing different device sensors. Next, we showcase different ways in which scripts send raw sensor data to remote servers. Lastly, we will look at the stability of our findings across different crawls taking place in the same geolocation and across different geolocations. US1 is our default dataset unless stated otherwise.

4.1 Prevalence of Scripts

First, we look at how often are device sensors accessed by scripts. Table 3 shows that sensor APIs are accessed on 3 695 of the 100K websites by scripts served from 603 distinct domains. Orientation and motion sensors are by far the most frequently accessed, on 2 653 and 2 036 sites respectively. This can be explained by common browser support for these APIs. Light and proximity sensors, which are only supported by Firefox, are accessed on fewer than 200 sites each.

Table 3: Overview of script access to sensor APIs. Columns indicate the number of sites and distinct script domains (i.e., domains from where scripts are served), respectively.

Sensor	Num. of sites	Num. of script domains
Motion	2653	384
Orientation	2036	420
Proximity	186	50
Light	181	35
Total	3695	603

We also look at the distribution of the sensor-accessing scripts among the Alexa top 100K sites. Figure 2 shows the distribution of the scripts across different ranked sites. Interestingly, we see that many of the sensor-accessing scripts are being served on top ranked websites. Table 4 gives a more detailed overview of the most common scripts that access sensor APIs. The scripts are represented by their Public Suffix + 1 (PS+1) addresses. In addition we calculated the *prominence* metric developed by Englehardt and Narayanan [31], which captures the rank of the different websites where a given script is loaded and sort the scripts according to this metric.

Table 4 shows that scripts from serving-sys.com, which belongs to advertising company Sizmek [75], access motion sensor data on 815 of the 100K sites crawled. Doubleverify, which has a very similar prominence score, provides advertising impression verification services [26] and has been known to use canvas fingerprinting [31]. The most prevalent scripts that access proximity and light sensors commonly belong to ad verification and fraud detection companies such as b2c.com and adsafeprotected.com. Both scripts also use battery and AudioContext API fingerprinting.

Although present on only 417 sites, alicdn.com script has the highest prominence score (0.3303) across all scripts. This is largely

Table 2: The list of high-level features and reference to methodology for detection.

High-level feature name	Description & Reference
audio_context_fingerprinting	Audio Context API fingerprinting via exploiting differences in the audio processing engine [31]
battery_fingerprinting	Battery status API fingerprinting via reading battery charge level and discharge time [62]
canvas_fingerprinting	Canvas fingerprinting via exploiting differences in the graphic rendering engine [1, 31]
canvas_font_fingerprinting	Canvas font fingerprinting via retrieving the list of supported fonts [31]
webrtc_fingerprinting	WebRTC fingerprinting via discovering public/local IP address [31]
easylist_blocked	Whether blocked by EasyList filter list [27]
easyprivacy_blocked	Whether blocked by EasyPrivacy filter list [28]
disconnect_blocked	Whether blocked by Disconnect filter list [25]

Table 4: Top script domains accessing device sensors sorted by prominence [31]. The scripts are grouped by domain to minimize over counting different scripts from each domain.

Sensor	Script Domain	Num. sites	Min. Rank	Prominence	EasyList blocked	EasyPrivacy blocked	Disconnect blocked
Motion	serving-sys.com	815	67	0.0485	0	1	1
	doubleverify.com	517	187	0.0453	1	0	0
	adsco.re	648	570	0.0275	1	0	0
Orientation	alicdn.com	417	9	0.3303	0	0	0
	adsco.re	648	570	0.0275	1	0	0
	yieldmo.com	83	100	0.0263	1	0	1
Proximity	b2c.com	108	498	0.0114	0	1	0
	adsafeprotected.com	36	1418	0.0023	1	0	1
	allrecipes.com	1	1216	0.0008	0	0	0
Light	b2c.com	108	498	0.0114	0	1	0
	adsafeprotected.com	36	1418	0.0023	1	0	1
	allrecipes.com	1	1216	0.0008	0	0	0

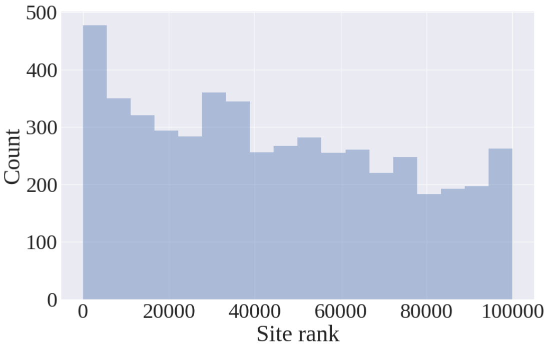

Figure 2: Distribution of sensor-accessing scripts across various ranked intervals.

Table 5: Number of sensor-accessing scripts served from first-party domains vs. third-party domains.

	Num. of first party	Num. of third party	Total
Motion	364	137	501
Orientation	350	300	650
Proximity	40	56	96
Light	30	52	82
Any sensor	518	398	916

because a script originating from alicdn.com accessed device orientation data on five of the top 100 sites—including taobao.com (Alexa global rank 9), the most popular site in our measurement where we detected sensor access—and thus this script is served to a very large user base. Table 5 shows the breakdown of sensor-accessing scripts in terms of first and third parties. While web measurement research commonly focuses on third-party tracking [50], we find that first-party scripts that access sensor APIs are slightly more common than third-party scripts. Our sensor exfiltration analysis of the scripts in section 4.2 revealed that many bot detection and mitigation scripts such as those provided by perimeterx.net and b2c.com are served from the clients' first party domains.

4.2 Sensor Data Exfiltration

After uncovering scripts that access device sensors, we investigate whether scripts are sending raw sensor data to remote servers. To accomplish this we spoof expected sensor values, as described in section 3.2. We then analyze HTTP request headers and POST request bodies obtained through OpenWPM's instrumentation to identify the presence of spoofed sensor values. We found several domains to access and send raw sensor data to remote servers either in clear text or in base64 encoded form.

Table 6 highlights the top ten script domains that send sensor data to remote servers. perimeterx.com (a bot detection company) and b2c.com (ad fraud detection company) are the most prevalent scripts that exfiltrate sensor readings. In addition, we found that priceline.com and kayak.com serve a copy of the perimeterx.com script from their domain (as a first-party script), which in turn reads and sends sensor data. These scripts send anywhere between one to tens of sensor readings to remote servers. Majority of the scripts (eight of ten) encode sensor data before sending it to a remote server. Appendix C lists examples of scripts sending sensor data to remote servers. We also found that certain scripts send statistical aggregates of sensor readings, and others obfuscate the code that is used to process sensor data and send it to a remote server. More examples are available in section 5.5.

While detecting exfiltration of spoofed sensor values, we use HTTP instrumentation data provided by OpenWPM. Since OpenWPM captures HTTP data in the browser (not on the wire, after it leaves the browser), our analysis was able to cover encrypted HTTPS data as well.

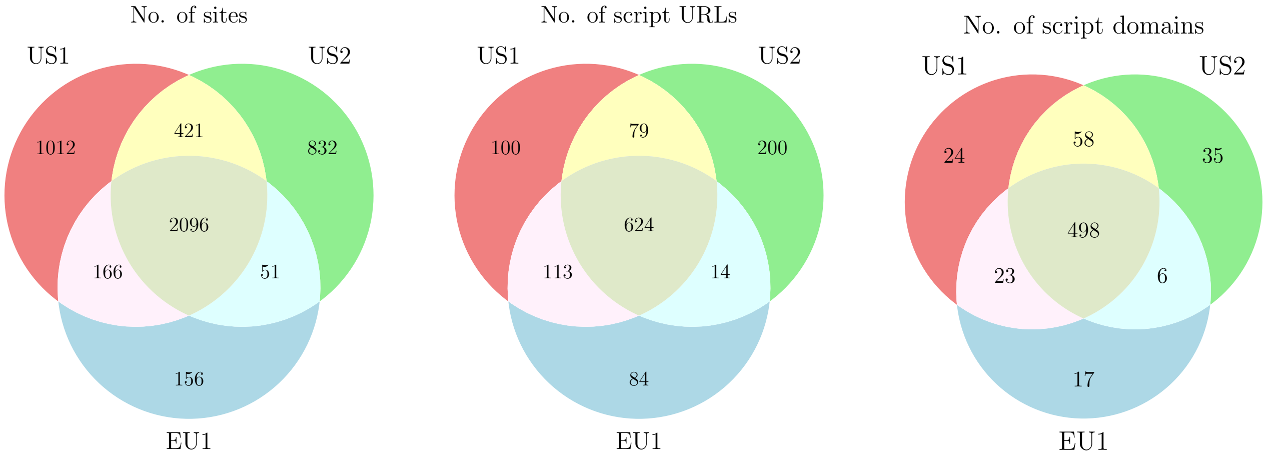

Figure 3: Overlap across different datasets.

Table 6: Domains of the scripts that send spoofed sensor data to remote servers.

Domain (PS+1)	Sensors*	Encoding	# of sites	Top site
b2c.com	A, O, P, L	base64	53	498
perimeterx.net	A	base64	45	247
wayfair.com	A	base64	7	1136
moatads.com	O	raw	5	3616
queit.in	A, O	raw	3	22935
kayak.com	A	base64	1	982
priceline.com	A	base64	1	1573
fiverr.com	A	base64	1	541
lulus.com	A	base64	1	4470
zazzle.com	A	base64	1	5860

* **'A'**: accelerometer, **'G'**: gyroscope, **'O'**: orientation, **'P'**: proximity, **'L'**: light

4.3 Crawl Comparison

In this section, we compare the results from our three data sets, US1, US2, and EU1. Figure 3 highlights the overlap and differences between the three crawls, presented as a Venn diagram. We conjecture that there are two main reasons for the observed differences between the results. First, most popular web sites are dynamic and change the ads, and sometimes the contents, that are displayed with each load. This is supported by the fact that although there are significant differences between the *sites* where sensors were accessed, the overlap between script URLs and domains is generally high.

Second, the location of the crawl appears to make a difference. The script domains in the two US crawls have more overlap (Jaccard index 0.86) than when comparing either US crawl to the one from the EU (Jaccard indices 0.79 and 0.83), even though all three were collected around the same time period. The absolute number of sites accessing sensors in the EU crawl was also smaller than in the US crawls by about a third (EU1: 2 469, US1: 3 695, US2: 3 400). It is possible that stricter privacy regulation in the EU, such as the EU's General Data Protection Regulation (GDPR) [32], may be responsible for this disparity, but we leave a full exploration of this question as future work.

5 UNDERSTANDING SENSOR USE CASES

Having identified scripts that access sensor APIs, we next focus on identifying the purpose of these scripts. To make this analysis tractable, we first use clustering to identify groups of similar scripts and then manually analyze the sensor uses cases.

5.1 Clustering Methodology

Clustering Process. In this section we will briefly describe the overall clustering process. We cluster JavaScript programs in three phases to generalize the clustering result as much as possible, and to accommodate for clustering errors that may have been caused by random noise introduced by the potentially varying behavior of scripts, such as incomplete page loads or intermittent crawler failures. Figure 4 highlights the three phases of the clustering process.

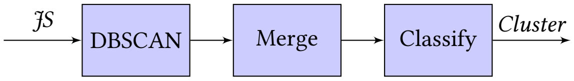

Figure 4: The three phases for clustering scripts.

In the first phase, we apply off-the-shelf DBSCAN [69], a density-based cluster algorithm, to generate the initial clusters, using the script features described in Section 3. In the second phase, we try to generalize the clustering results by merging clusters that are similar. We do this in an iterative manner where in each round we determine the pair of clusters, merging which would result in the least amount of reduction in the average *silhouette coefficient*.[5] This process is repeated until any new merges would reduce the average silhouette coefficient reduced by more than a given threshold (δ).

In the last phase, we try to see if certain samples that are categorized as noisy can be classified into one of the other core clusters. The reason behind this step is to see if certain scripts were incorrectly clustered due to differences in their behavior across different

[5] Here, we only consider clusters that are not labeled as noisy

websites. The same script may exhibit a different behavior when publishers (first parties) use different features of the script, or when the script execution depends on the loading of dynamic content such as ads. To perform classification we use a random forest classifier [70], where the non-noisy cluster samples, labeled with their corresponding cluster label, serve as the training data. We then try to classify the noisy samples as members of one of the core clusters. We relabel the scripts only if the prediction probability by the classifier is greater than a given threshold (θ). Pseudo-code (in Python) for the three phases is provided in appendix A.

Validation Methodology. To validate our clustering results and to determine the different use cases for accessing sensor data we take the following two steps: First, we generate an average similarity score per cluster by computing the pairwise code difference between two scripts using the Moss tool [3]. Next, if the average similarity score for a given cluster is lower than a certain threshold (ϵ), we manually analyze five random scripts from that cluster, otherwise (if higher than the threshold) we manually analyze three random scripts per cluster.

For manual analysis we follow a protocol of steps given below:

- Inspect the code description, copyright statements, software license, links to public repositories, if any, to search for any stated purpose of the script.
- Statically analyze the registered sensor event listeners to determine how sensor data is used.
- If static analysis fails (e.g., due to obfuscation), load a page that embeds the script and debug over the USB, using Chrome developer tools with break points enabled for sensors event listeners, to analyze runtime behavior.
- Check if sensor data is leaving the browser, i.e., if the script makes any HTTP/HTTPS requests containing sensor data.
- If the script sends some encoded data, try decoding the payload.

5.2 Clustering Scripts

We next describe the detailed process and results of clustering the scripts. We first try to cluster scripts based on low-level features described in section 3.4. Recall that low-level features include browser properties accessed (by either *get* or *set* method) and function calls made (using call or addEventListener) by the script. The reason behind the use of low-level features is that it provides us with a comprehensive overview of the script's functionality. We start by only considering scripts that access any of the four sensors we study: motion, orientation, proximity or light. We found 916 such scripts in our US1 dataset. Next, we cluster these scripts using DBSCAN [69]. Figure 5 highlights the clustering results where the x axis represents the silhouette coefficient per cluster. We see that there are 39 distinct clusters generated by DBSCAN, of which around 24% scripts are labeled as noisy (i.e., scripts that are labeled as '-1'). The red and blue vertical lines in the figure present the average silhouette coefficient with and without the noisy samples, respectively.

In order to generalize our clustering results we attempt to merge similar clusters, i.e., clusters that result in the least amount of reduction in silhouette coefficient when merged (see appendix A for code). We set the total reduction in silhouette coefficient to 0.01 (i.e., $\delta = 0.01$). Doing so reduces the total number of clusters to 36 but certain clusters such as cluster number 37 becomes a bit more noisy. Figure 6 highlight the merged clustering results.

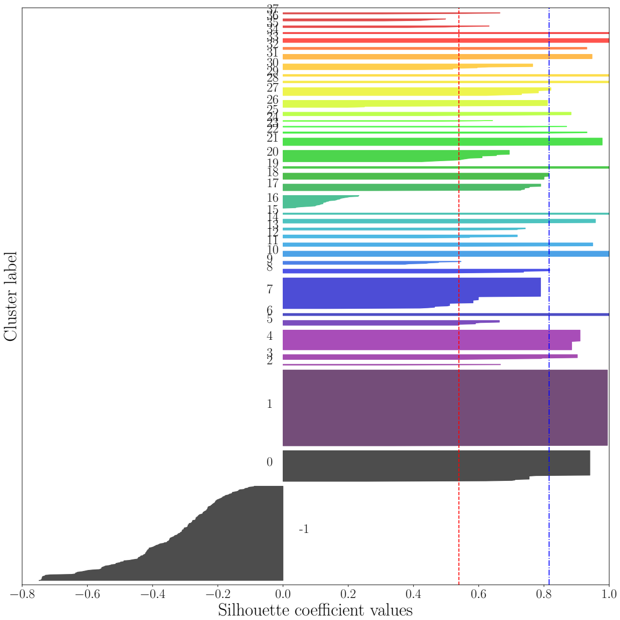

Figure 5: Clustering scripts based on low-level features.

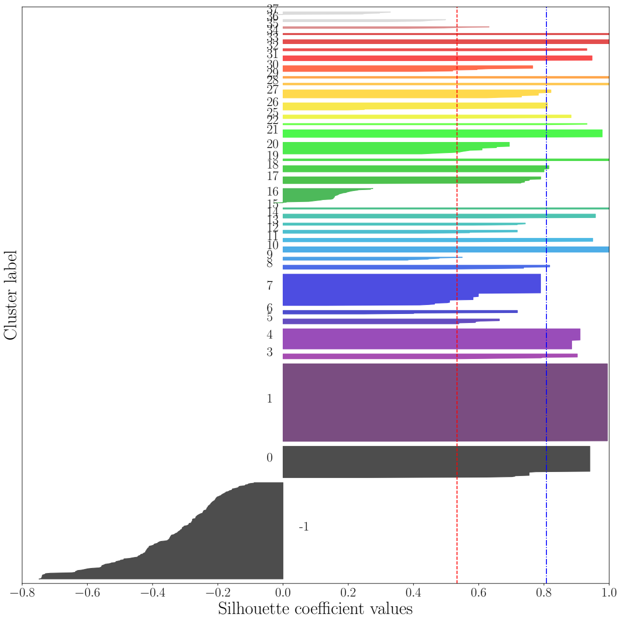

Figure 6: Merging similar clusters until average silhouette coefficient reduce by 0.01.

Finally, we check if certain noisy samples (i.e., scripts that are labeled as '-1') can be classified into one of the other core clusters with a certain probability. To do this we use a random forest classifier, where scripts from non-noisy clusters (i.e., clusters that are labeled with a value ≥ 0) are used as training data and scripts that are noisy are used as testing data. We only relabel noisy samples if the prediction probability, $\theta \geq 0.7$.[6] Also we update labels in batches of five samples at a time. Figure 7 highlights the outcome of this final step. This reduces the total fraction of scripts labeled as noisy from 24% to 21%. However, as evident from Figure 7 this also increases the chance of certain clusters becoming slightly more noisy (e.g., clusters 16). The average silhouette coefficient without the remaining noisy samples (i.e., ignoring the cluster labeled as '-1') after this phase is close to 0.8 which is an indication that the clustering outcomes are within an acceptable range. For understanding the impact geo-location, we also ran our clustering techniques on the EU1 dataset and obtained similar results. We found a total of 46 clusters with approximately 23% of the scripts labeled as noisy. In section 5.4, we will briefly discuss how in spite of the total number of clusters being larger compared to the US1 dataset they represent similar use cases.

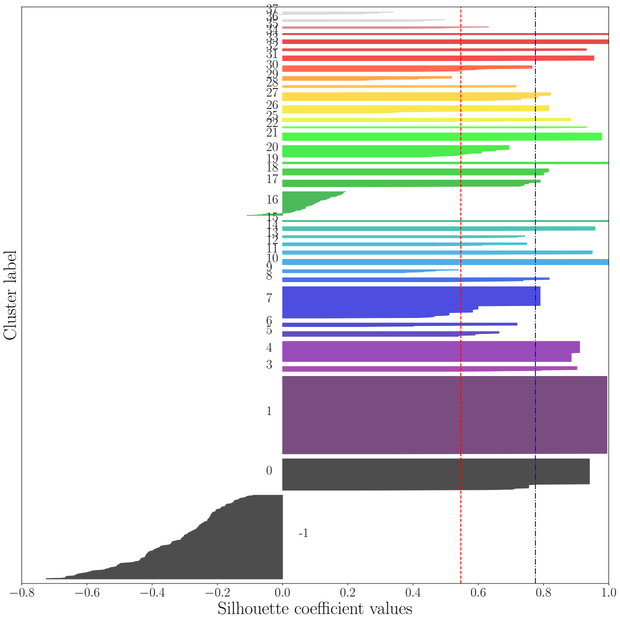

Figure 7: Classifying noisy samples using non-noisy samples as ground truth, only if prediction probability ≥ 0.7.

5.3 Validating Clustering Results

We use the Moss [3] service, which measures source code similarity to detect plagiarism, in order to validate the results of our clustering. We use Moss to calculate the similarity scores between pairs of scripts from the same cluster. For comparison, we also calculate the similarity between pairs of scripts from different clusters; in this case, we limit ourselves to five random scripts per cluster due to the rate limitations imposed by Moss.

We plot the distribution of these scores in Figure 8. We note that scripts within the same cluster tend to have high similarity; in particular, 81% of pairs have a similarity score exceeding 0.7. Likewise, scripts from different clusters tend to be dissimilar, with 94% of samples showing a similarity score of 0.1 or less. This suggests that the clusters are identifying groups of scripts that have high source-level similarity.

We also compute the average pairwise similarity score for each cluster to guide our manual analysis. For clusters with high average pairwise similarity scores ($\epsilon > 0.7$), we manually inspect three randomly-chosen scripts from each cluster. For clusters with lower similarity scores, we inspect five random scripts per cluster.[7]

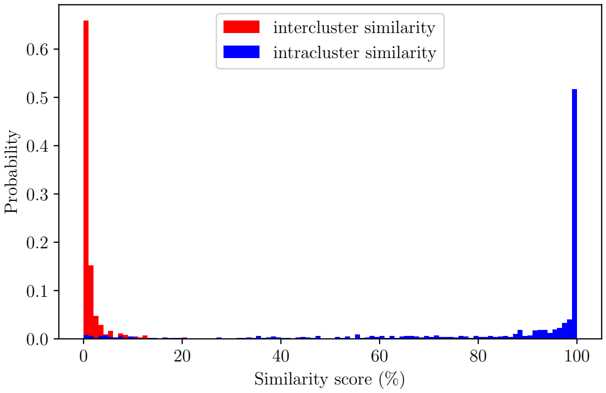

Figure 8: Distribution of intra- and inter-cluster similarity.

5.4 Real-world Use Cases

Table 7 summarizes the different use cases that we have identified through our manual inspection. The table also highlights the average pairwise code similarity score per cluster, computed through Moss [3]. It should be noted that the high similarity scores likely result from sites using or copying code from common libraries, whereas the low scores result from scenarios where only small parts of the scripts were either reused or copied from other scripts. We see that there are broadly seven different use cases for accessing sensor data, among which around 37% of the scripts collect sensor data to perform tracking and analytics such as audience recognition, ad impression verification, and session replay. We also see that around 18% scripts use sensor data to determine whether a device is a smartphone or a bot to deter fraud. Interestingly, 70% and 76% of the scripts described in these two categories, respectively, have been identified to be doing some combination of canvas, webrtc, audio_context or battery fingerprinting.

We found similar use cases for the EU1 dataset. Around 19% of the scripts were found to use sensor data to distinguish bots from real smartphones. We did, however, see somewhat a lower percentage of scripts (around 31%) involved in tracking and analytics. We found this group of scripts to be loaded on only 330 sites whereas for the US1 dataset this number was more than three times bigger (1198).

[6] This value was empirically set by manually spot checking how effective the classification results were.

[7] For any clusters with five scripts or fewer, we manually inspect all the scripts.

Table 7: Potential sensor access use cases.

Cluster ID*	% of JS	# of Sites	# of Sites Ranked ...			Avg. Sim.(%) per cluster	Description
			1–1K	1K–10K	10K–100K		
34	0.4	4	0	0	4	99	Use sensor data to add entropy to random numbers [77]
7, 19, 20	11.2	114	2	32	80	89, 91, 43	Checks what HTML5 features are offered [22, 24, 54]
0, **3**, **5**, 12, **18** **22**, **25**, 27, 31	17.7	413	10	53	350	93, 69, 23, 95, 99 92, 36, 81, 83	Differentiating bots from real devices [33, 64]
26, 30	3.4	35	1	2	32	37, 60	Parallax Engine that reacts to orientation sensors [86]
6, 14, 33	3.2	103	0	9	94	97, 98, 99	Automatically resize contents in page or iframe [10]
8, 11, 21, 28, 29, **32**	6.7	533	18	118	397	99, 96, 99, 11, 43, 89	Reacting to orientation, tilt, shake [35, 42, 45]
1, **4**, 9, 10, 13, 15 16, **17**, 35, 36, 37	36.8	1198	24	144	1030	99, 68, 98, 99, 92, 99 91, 91, 38, 87, 40	Tracking, analytics, fingerprinting and audience recognition [34, 75]
-1	20.6	1804	16	176	1612	4	Scripts clustered as noisy

* Clusters with bolded IDs consist of scripts that have been identified as performing some combination of canvas, webrtc, audio_context or battery fingerprinting

5.5 Analysis of Specific Scripts

While manually analyzing scripts for clustering validation, we uncovered interesting uses of the sensor data. Here we will briefly discuss two such scripts. The first script comes from doubleverify.com [26], an ad impression verification company. One of their scripts computes statistical features such as average and variance of motion sensor data before sending it to their remote server. Such statistical features have been shown to be useful for fingerprinting smartphones [19]. The code segment is provided in appendix B (Listing 2). Since this script was sending statistical data instead of raw sensor data it was not captured through our sensor spoofing mechanism. We were only able to identify this via manually debugging the script on a real smartphone (through USB debugging features of the Chrome Devtool). We found doubleverify.com scripts being loaded on 517 websites of which 7 appeared in the Alexa top 1000 sites (in our US1 dataset). However, since doubleverify.com evaluates ad impressions the presence of these scripts is dependent on the ads that are served on a website, and hence we see it on different sites in different crawls. For instance, in US2 dataset we found 509 sites loading the script from doubleverify.com. The union of these datasets results in 881 unique sites loading the script, of which 145 sites were common (Jaccard index 0.16). In the EU1 dataset (European crawl), however, doubleverify.com was not present on any of the 100K sites, indicating that the loading of scripts may depend on the location of the visitor.

Some sensor reading scripts are served from the first party's domain, making the attribution to specific providers more difficult. For instance, a highly obfuscated script that is present on popular sites like homedepot.com and staples.com is always served on the /_bm/async.js path under the first party (e.g., m.staples.com/_bm/async.js). This script sends encoded sensor data in a POST request to the endpoint _bm/_data on the first-party site. A code snippet is provided in appendix B (Listing 4). The prevalence of these scripts was more or less similar as it is site dependent rather than being ad dependent. We found 173 sites loading such scripts in the US1 dataset, 12 of which were ranked in the Alexa top 1000 sites. For the US2 and EU1 dataset we found 140 and 158 sites loading such scripts, respectively.

6 EFFICACY OF COUNTERMEASURES

In this section we study the overlap between scripts that access sensors and scripts that perform fingerprinting. We then study the effectiveness of privacy countermeasures such as ad blocking lists as well as browser limitations on sensor APIs.

6.1 Fingerprinting Scripts

First, we will showcase to what extent scripts accessing sensor APIs overlap with fingerprinting scripts. To detect fingerprinting scripts we follow methodologies from existing literatures [1, 31], which are also listed in Table 2. Table 8 highlights the percentage of sensor accessing scripts that also utilize browser fingerprinting, as captured by the features described in section 3.4. We calculate the percentage of scripts accessing a given sensor that also perform a particular type of fingerprinting. For example, 62.7% of scripts that access motion sensors also engage in some form of browser fingerprinting.

Table 8: Percentage of sensor accessing scripts that also engage in fingerprinting. All columns except 'Total' are given as percentage. The 'Total' column shows the number of distinct script URLs that access a certain sensor.

	Canvas FP	Canvas Font FP	Audio FP	WebRTC FP	Battery FP	Any FP	Total
Motion	56.7	0.2	19.8	6.8	5.6	62.7	501
Orientation	36.2	3.4	5.7	6.2	4.5	41.7	650
Proximity	2.1	0.0	47.9	0.0	49.0	51.0	96
Light	19.5	1.2	56.1	15.9	57.3	76.8	82

Table 9 showcases the numbers from the other angle: the fraction of fingerprinting scripts that access different sensor APIs. We list the percentage of distinct script URLs that engage in a particular form of fingerprinting while accessing any of the sensors explored in our study. Both of these tables indicate that there is a significant overlap between the fingerprinting scripts and the scripts accessing sensor APIs.

6.2 Ad Blocking and Tracking Protection Lists

We next inspect what fraction of these sensor-accessing scripts would be blocked by different well known filtering lists used for ad blocking and tracking protection: EasyList [27], EasyPrivacy [28]

Table 9: Percentage of fingerprinting scripts that also access sensors. All columns except 'Total' are given as percentage. The 'Total' column shows the number of distinct fingerprinting script URLs that use a particular fingerprinting method.

	Motion	Orientation	Proximity	Light	Any sensor	Total
Canvas FP	1.4	1.5	2.0	2.0	15.9	1991
Canvas Font FP	32.9	34.1	47.1	47.1	24.7	85
Audio FP	20.0	20.7	28.6	28.6	81.4	140
WebRTC FP	10.5	10.9	15.0	15.0	20.2	267
Battery FP	4.5	4.6	6.4	6.4	7.5	625

and Disconnect [25]. Table 10 highlight the percentage of tracking scripts that would be blocked using each of these lists. In general, we see that a significant portion of scripts that access sensors are missed by the popular blacklists, which is in line with the previous research on tracking protection lists [52].

Table 10: Percentage of script domains accessing device sensors that are blocked by different filtering list.

Sensor	Disconnect blocked	EasyList blocked	EasyPrivacy blocked
Motion	1.8%	1.8%	2.9%
Orientation	3.6%	3.1%	3.1%
Proximity	6.0%	2.0%	4.0%
Light	2.9%	2.9%	8.6%
Any sensor	2.9%	2.5%	3.3%

6.3 Difference in Browser Behavior

To determine browser support for different sensor APIs and potential restrictions for scripts in cross-origin iframes, we set up a test page that accesses all four sensor APIs. We tested the latest version of nine browsers listed in Table 11 as of Jan, 2018. Browsers have minor differences with regards to which sensor they support and how they block access from scripts embedded in cross-origin iframes. Table 11 summarizes our findings. As shown in the table, proximity and light sensors are only supported by Firefox.[8] For privacy reasons, Firefox and Safari do not allow scripts from cross-origin iframes to access sensor data, which is in line with W3C recommendation [80]. Privacy-geared browsers such as Firefox Focus and Brave fare worse than Firefox and Safari, as they both allow access to orientation data from cross-origin iframes, where Firefox Focus further allows access to motion data.

Testing the sensor API availability on insecure (HTTP) pages, we found no differences in browsers' behavior. We also tested whether browsers have any access restrictions when running in private browsing mode, and we found no difference when comparing to normal browsing mode.[9] Finally, to test whether the underlying mobile platform has any effect on sensor availability, we tested iOS versions of the browsers. We found that all browsers behave identical to Safari, as Apple requires browsers to use WebKit framework to be listed on their app store [21].

Table 11: Browser support for different sensor APIs.

Browser	Orientation*	Motion*	Proximity*	Light*
Chrome	(✓, ✓)	(✓, ✓)	(✗, ✗)	(✗, ✗)
Edge	(✓, ✓)	(✓, ✓)	(✗, ✗)	(✗, ✗)
Safari	(✓, ✗)	(✓, ✗)	(✗, ✗)	(✗, ✗)
Firefox	(✓, ✗)	(✓, ✗)	(✓, ✗)	(✓, ✗)
Brave	(✓, ✓)	(✓, ✗)	(✗, ✗)	(✗, ✗)
Focus	(✓, ✓)	(✓, ✓)	(✗, ✗)	(✗, ✗)
Dolphin	(✓, ✓)	(✓, ✓)	(✗, ✗)	(✗, ✗)
Opera Mini	(✓, ✓)	(✓, ✓)	(✗, ✗)	(✗, ✗)
UC Browser	(✓, ✓)	(✓, ✓)	(✗, ✗)	(✗, ✗)

* Each tuple representing (third-party, iframe) access right

We filed bug reports for Brave Android Browser, Firefox Focus and Firefox for Android [12, 36–38] pointing out that they allow sensor access on insecure pages, which is against W3C recommendations. Firefox Focus engineers told us that they will have to wait for Chromium/WebView to ship an update for this behavior to change since Firefox Focus on Android uses Chromium under the hood. Responding to the issue we filed for Firefox for Android, Mozilla engineers briefly discussed the possibility of requiring user permission for allowing sensor access. We did not get any response to our issue from Brave engineers. We note that Brave Android is also built on Chromium.

7 DISCUSSION AND RECOMMENDATIONS

Our analysis of crawling the Alexa top 100K sites indicates that tracking scripts did not wait long to take advantage of sensor data, something that is easily accessible without requiring any user permission. By spoofing real sensor values we found that third-party ad and analytics scripts are sending raw sensor data to remote servers. Moreover, given that existing countermeasures for mobile platforms are not effective at blocking trackers, we make the following recommendations.

- W3C's recommendation for disabling sensor access on cross-origin iframes [80] will limit the access from untrusted third-party scripts and is a step in the right direction. However, Safari and Firefox are the only two browsers that follow this recommendation. Our measurements indicate that scripts that access sensor APIs are frequently embedded in cross-origin iframes (67.4% of the 31 444 cases). This shows that W3C's mitigation would be effective at curbing the exposure to untrusted scripts. Allowing sensor access on insecure pages is another issue where browsers do not follow the W3C spec: all nine browsers we studied allowed access to sensors on insecure (HTTP) pages.
- Feature Policy API [16], if deployed, will allow publishers to selectively disable JavaScript APIs. Publisher may disable sensor APIs using this API to prevent potential misuses by the third-party scripts they embed.

[8] As of May 9, 2018 Mozilla released Firefox version 60, which disables proximity and light sensor APIs; we used an earlier version of Firefox in our study.

[9] Note that Firefox Focus always runs in private browsing mode, so it does not have a separate normal browsing mode.

- Provide low resolution sensor data by default, and require user permission for higher resolution sensor data.
- To improve user awareness and curb surreptitious sensor access, provide users with a visual indication that the sensor data is being accessed.
- Require user permission to access sensor data in private browsing mode, limit resolution, or disable sensor access all together.

8 LIMITATIONS

Our clustering analysis depends on OpenWPM's instrumentation data to attribute JavaScript behavior to individual scripts. There are potential imperfections in this attribution task. First, some websites concatenate several JavaScript files and libraries into a single file. These scripts would be seen as one script (URL) to OpenWPM's instrumentation, potentially adding noise in the clustering stage. Second, when attributing JavaScript function calls and property accesses to individual scripts, we use the script URL that appears at the top of the calling stack following the prior work done by Englehardt and Narayanan [31]. Under some circumstances, this approach may be misleading. For instance, when a script uses jQuery library to listen to sensor events, we attribute the sensor related feature to jQuery as it appears at the top of the calling stack.

OpenWPM-Mobile uses OpenWPM's JavaScript instrumentation, which captures function calls made and browser properties accessed at runtime (Section 3.4). This approach has the advantage of capturing the behavior of obfuscated code, but may miss code segments that do not execute during a page visit.

We manually analyzed a random subsample of scripts instead of studying all scripts per cluster. While this process may miss some misbehaving scripts, we believe the outcomes will not be affected as the average intra- and inter-cluster similarity scores are significantly apart.

OpenWPM-Mobile does not store in-line scripts. We found that only 12.1% (111 of 916) of the scripts were in-line. We were able to re-crawl sites that included the in-line scripts and stored them for the clustering step.

There are many ways in which trackers can exfiltrate sensor data, for example, using encryption or computing and sending statistics on the sensor data as we present in section 5.5. Therefore, our results on sensor data exfiltration should be taken as lower bounds.

Using fingerprinting test suites fingerprintjs2 [84] and EFF's Panopticlick [30], we verified that OpenWPM-Mobile's browser fingerprint matches that of a Firefox for Android running on a real smartphone to the best extent possible. We also observed several ad platforms identify our browser as mobile and start serving mobile ads. However, there may still be ways, for example, detecting the lack of hand movements in the sensor data stream could potentially help websites detect OpenWPM-Mobile as an automated desktop browser and treat differently.

9 CONCLUSION

Our large-scale measurement of sensor API usage on the mobile web reveals that device sensors are being used for purposes other than what W3C standardization body had intended. We found that a vast majority of third-party scripts are accessing sensor data for measuring ad interactions, verifying ad impressions, and tracking devices. Our analysis uncovered several scripts that are sending raw sensor data to remote servers. While it is not possible to determine the exact purpose of this sensor data exfiltration, many of these scripts engage in tracking or web analytic services. We also found that existing countermeasures such as Disconnect, EasyList and EasyPrivacy were not effective at blocking such tracking scripts. Our evaluation of nine popular mobile browsers has shown that browsers, including the privacy-oriented Firefox Focus and Brave, commonly fail to implement the mitigation guidelines recommended by the W3C against the misuse of sensor data. Based on our findings, we recommend browser vendors to rethink the risks of exposing sensitive sensors without any form of access control mechanism in place. Also, website owners should be given more options to limit the sensor misuse from untrusted third-party scripts.

10 ACKNOWLEDGEMENTS

We would like to thank all the anonymous reviewers for their feedback. We would also like to thank Arvind Narayanan, Steven Englehardt and our shepherd Ben Stock for their valuable feedback. This material is based in part upon work supported by the National Science Foundation under Grant No. 1739966.

REFERENCES

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The Web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 674–689.

[2] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 20th ACM SIGSAC conference on Computer and Communications Security (CCS)*. 1129–1140.

[3] Alex Aiken. 2018. A system for detecting software similarity. https://theory.stanford.edu/~aiken/moss/.

[4] Furkan Alaca and PC van Oorschot. 2016. Device fingerprinting for augmenting web authentication: Classification and analysis of methods. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 289–301.

[5] Alexa 2018. Alexa top sites service. https://www.alexa.com/topsites.

[6] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. 2009. SurroundSense: Mobile phone localization via ambience fingerprinting. In *Proceedings of the 15th annual international conference on Mobile computing and networking*. 261–272.

[7] T. Berners-Lee, R. Fielding, and L. Masinter. 2005. RFC 3986, Uniform Resource Identifier (URI): Generic syntax. http://www.ietf.org/rfc/rfc3986.txt.

[8] Frédéric Besson, Nataliia Bielova, and Thomas Jensen. 2014. Browser randomisation against fingerprinting: A quantitative information flow approach. In *Nordic Conference on Secure IT Systems*. 181–196.

[9] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. 2014. Mobile device identification via sensor fingerprinting. *CoRR* abs/1408.1416 (2014). http://arxiv.org/abs/1408.1416

[10] David J. Bradshaw. 2017. iFrame resizer. https://github.com/davidjbradshaw/iframe-resizer.

[11] Brave Browser 2018. Fingerprinting protection mode. https://github.com/brave/browser-laptop/wiki/Fingerprinting-Protection-Mode.

[12] Bugzilla. 2018. 1436874 - Restrict device motion and orientation events to secure contexts. https://bugzilla.mozilla.org/show_bug.cgi?id=1436874.

[13] Elie Bursztein, Artem Malyshev, Tadek Pietraszek, and Kurt Thomas. 2016. Picasso: Lightweight device class fingerprinting for web clients. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*. 93–102.

[14] Liang Cai and Hao Chen. 2012. On the practicality of motion based keystroke inference attack. In *International Conference on Trust and Trustworthy Computing*. Springer, 273–290.

[15] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser fingerprinting via OS and hardware level features. In *Proceeding of 24th Annual Network and Distributed System Security Symposium (NDSS)*.

[16] Ian Clelland. 2017. Feature policy: Draft community group report. https://wicg.github.io/feature-policy/.

[17] Anupam Das, Gunes Acar, and Nikita Borisov. 2018. A Crawl of the mobile web measuring sensor accesses. University of Illinois at Urbana-Champaign. https://doi.org/10.13012/B2IDB-9213932_V1

[18] Anupam Das, Nikita Borisov, and Matthew Caesar. 2014. Do you hear what I hear?: Fingerprinting smart devices through embedded acoustic components. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 441–452.

[19] Anupam Das, Nikita Borisov, and Matthew Caesar. 2016. Tracking mobile web users through motion sensors: Attacks and defenses. In *Proceeding of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*.

[20] Anupam Das, Nikita Borisov, and Edward Chou. 2018. Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures. *Proceedings on the 18th Privacy Enhancing Technologies (PoPETs)* 1 (2018), 88–108.

[21] Apple developers. 2018. App store review guidelines. https://developer.apple.com/app-store/review/guidelines/.

[22] DeviceAtlas 2018. Device browser. https://deviceatlas.com/device-data/devices.

[23] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. 2014. AccelPrint: Imperfections of accelerometers make smartphones trackable. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*.

[24] Digioh 2018. We give marketers power. http://digioh.com/.

[25] Disconnect 2018. Disconnect defends the digital you. https://disconnect.me/.

[26] DoubleVerify 2018. Authentic impression. https://www.doubleverify.com/.

[27] EasyList authors 2018. EasyList. https://easylist.to/easylist/easylist.txt.

[28] EasyPrivacy authors 2018. EasyPrivacy. https://easylist.to/easylist/easyprivacy.txt.

[29] Peter Eckersley. 2010. How unique is your web browser?. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies (PETS)*. 1–18.

[30] Electronic Frontier Foundation 2018. Panopticlick. https://panopticlick.eff.org/.

[31] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[32] European Commission 2018. The General Data Protection Regulation (GDPR). https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_en.

[33] F5 2018. Silverline web application firewall. https://f5.com/products/deployment-methods/silverline/cloud-based-web-application-firewall-waf.

[34] ForeSee 2018. CX with certainty | ForeSee. https://www.foresee.com/.

[35] Alex Gibson. 2015. Detecting shake in mobile device. https://github.com/alexgibson/shake.js/.

[36] GitHub. 2018. Disallow sensor access on insecure contexts. https://github.com/brave/browser-android-tabs/issues/549.

[37] GitHub. 2018. Disallow sensor access on insecure contexts. https://github.com/mozilla-mobile/focus-android/issues/2092.

[38] GitHub. 2018. Firefox Focus is making sensor APIs available to cross-origin iFrames. https://github.com/mozilla-mobile/focus-android/issues/2044.

[39] Jun Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang. 2012. ACComplice: Location inference using accelerometers on smartphones. In *Proceedings of the 4th International Conference on Communication Systems and Networks (COMSNETS)*. 1–9.

[40] J. Hua, Z. Shen, and S. Zhong. 2017. We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones. *IEEE Transactions on Information Forensics and Security* 12, 2 (2017), 286–297.

[41] Thomas Hupperich, Davide Maiorca, Marc Kührer, Thorsten Holz, and Giorgio Giacinto. 2015. On the robustness of mobile device fingerprinting: Can mobile users escape modern web-tracking mechanisms?. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC)*. ACM, 191–200.

[42] jQuery Mobile 2018. Orientationchange event. https://api.jquerymobile.com/orientationchange/.

[43] Jonathan Kingston. 2018. Bug 1359076 - Disable devicelight, deviceproximity and userproximity events. https://bugzilla.mozilla.org/show_bug.cgi?format=default&id=1359076.

[44] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. 2005. Remote physical device fingerprinting. *IEEE Transaction on Dependable Secure Computing* 2, 2 (2005), 93–108.

[45] Oleg Korsunsky. 2018. Polyfill for CSS position: sticky. https://github.com/wilddeer/stickyfill.

[46] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. 2017. Fingerprinting mobile devices using personalized configurations. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2016, 1 (2017), 4–19.

[47] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *International Symposium on Engineering Secure Software and Systems*. Springer, 97–114.

[48] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P)*. 878–894.

[49] Jonathan R Mayer. 2009. "Any person... a pamphleteer": Internet anonymity in the age of Web 2.0. *Undergraduate Senior Thesis, Princeton University* (2009).

[50] Jonathan R Mayer and John C Mitchell. 2012. Third-party web tracking: Policy and technology. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)*. 413–427.

[51] Maryam Mehrnezhad, Ehsan Toreini, Siamak F. Shahandashti, and Feng Hao. 2015. TouchSignatures: Identification of user touch actions based on mobile sensors via JavaScript. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. 673–673.

[52] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. 2017. Block me if you can: A large-scale study of tracker-blocking tools. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. 319–333.

[53] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. 2014. Gyrophone: Recognizing speech from gyroscope signals. In *Proceedings of the 23rd USENIX Conference on Security Symposium*. 1053–1067.

[54] Modernizr. 2018. Respond to your user's browser features. https://modernizr.com/.

[55] Sue B. Moon, Paul Skelly, and Don Towsley. 1999. Estimation and removal of clock skew from network delay measurements. In *Proceedings of the 18th Annual IEEE International Conference on Computer Communications (INFOCOM)*. 227–234.

[56] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of Web 2.0 Security and Privacy Workshop (W2SP)*.

[57] Mozilla Foundation 2018. Public suffix list. https://publicsuffix.org/.

[58] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are what you include: Large-scale evaluation of remote JavaScript inclusions. In *Proceedings of the 19th ACM SIGSAC conference on Computer and Communications Security (CCS)*. 736–747.

[59] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. PriVaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. 820–830.

[60] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P)*. 541–555.

[61] Lukasz Olejnik. 2017. Stealing sensitive browser data with the W3C ambient light sensor API. https://blog.lukaszolejnik.com/stealing-sensitive-browser-data-with-the-w3c-ambient-light-sensor-api/.

[62] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. 2015. The leaking battery. In *International Workshop on Data Privacy Management*. 254–263.

[63] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. 2012. ACCessory: Password inference using accelerometers on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile)*. 9:1–9:6.

[64] PerimeterX 2018. Stop bot attacks: Bot detection and bot protection with unparalleled accuracy. https://www.perimeterx.com/.

[65] Mike Perry, Erinn Clark, Steven Murdoch, and George Koppen. 2018. The design and implementation of the Tor browser (DRAFT). https://www.torproject.org/projects/torbrowser/design/.

[66] Davy Preuveneers and Wouter Joosen. 2015. Smartauth: Dynamic context fingerprinting for continuous user authentication. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 2185–2191.

[67] Samsung Newsroom 2014. 10 Sensors of Galaxy S5: Heart rate, finger scanner and more. https://news.samsung.com/global/10-sensors-of-galaxy-s5-heart-rate-finger-scanner-and-more.

[68] Florian Scholz et al. 2017. Devicemotion - web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/Events/devicemotion.

[69] scikit-learn developers 2018. Python sklearn DBSCAN. http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

[70] scikit-learn developers 2018. Python sklearn RandomForestClassifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[71] Connor Shea et al. 2018. DeviceLightEvent - web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/DeviceLightEvent.

[72] Connor Shea et al. 2018. DeviceProximityEvent - web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/DeviceProximityEvent.

[73] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul J. M. Havinga. 2014. Fusion of smartphone motion sensors for physical activity recognition. *Sensors* 14, 6 (2014), 10146–10176.

[74] Ronnie Simpson. 2016. Mobile and tablet internet usage exceeds desktop for first time worldwide | StatCounter Global Stats. http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide.

[75] Sizmek 2018. Impressions that inspire. https://www.sizmek.com/.

[76] Jan Spooren, Davy Preuveneers, and Wouter Joosen. 2015. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proceedings of the 8th European Workshop on System Security (EuroSec)*. ACM, 1–6.

[77] Emily Stark, Mike Hamburg, and Dan Boneh. 2017. Stanford JavaScript Crypto Library. https://github.com/bitwiseshiftleft/sjcl/blob/master/sjcl.js.

[78] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the fingerprintability of browser extensions. In *Proceeding of the 38th IEEE Symposium on Security and Privacy (S&P)*. 941–956.

[79] Xing Su, Hanghang Tong, and Ping Ji. 2014. Activity recognition with smartphone sensors. *Tsinghua Science and Technology* 19, 3 (2014), 235–249.

[80] Rich Tibbett, Tim Volodine, Steve Block, and Andrei Popescu. 2018. Device-Orientation event specification. https://w3c.github.io/deviceorientation/

[81] Christof Ferreira Torres, Hugo Jonker, and Sjouke Mauw. 2015. FP-Block: Usable web privacy by controlling browser fingerprinting. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 3–19.

[82] Tom Van Goethem and Wouter Joosen. 2017. One side-channel to bring them all and in the darkness bind them: Associating isolated browsing sessions. In *Proceeding of the 11th USENIX Workshop on Offensive Technologies (WOOT)*.

[83] Tom Van Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. 2016. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In *Proceeding of the International Symposium on Engineering Secure Software and Systems*. 106–121.

[84] Valentin Vasilyev. 2018. fingerprintjs2: Modern & flexible browser fingerprinting library. https://github.com/Valve/fingerprintjs2.

[85] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking browser fingerprint evolutions. In *Proceeding of the 39th IEEE Symposium on Security and Privacy (S&P)*. 1–14.

[86] Matthew Wagerfield. 2017. Parallax.js. https://github.com/wagerfield/parallax.

[87] Rick Waldron, Mikhail Pozdnyakov, and Alexander Shalamov. 2017. Sensor use cases: W3C Note. https://w3c.github.io/sensors/usecases.html.

[88] Zhi Xu, Kun Bai, and Sencun Zhu. 2012. TapLogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*. 113–124.

[89] Zhe Zhou, Wenrui Diao, Xiangyu Liu, and Kehuan Zhang. 2014. Acoustic fingerprinting revisited: Generate stable device ID stealthily with inaudible sound. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 429–440.

A CLUSTERING PSEUDO-CODE

```
# check if clusters can be combined
def pairwise_cluster_comparison(features, clusters):
    pairwise_merge = {}
    for i in sorted(set(clusters)):
        for j in sorted(set(clusters)):
            if i == j or i == -1 or j == -1: continue
            labels = np.copy(clusters) # restore original labels
            labels[labels == j] = i
            inds = np.where(labels >= 0)[0] # only consider non-noisy samples
            val = silhouette_score(features[inds,:], labels[inds])
            pairwise_merge[(i,j)] = val
    return pairwise_merge
# classify noisy samples
def classification(features, labels, thres, limit=None):
    clusters = np.copy(labels)
    clf = RandomForestClassifier(n_estimators=100, max_features='auto')
    X_train = features[np.where(labels >= 0), :]
    y_train = labels[np.where(labels >= 0), :]
    X_test = features[np.where(labels == -1), :]
    y_test = labels[np.where(labels == -1), :]
    clf.fit(X_train, y_train)
    res = clf.predict(X_test)
    prob = clf.predict_proba(X_test)
    max_prob = np.max(prob, axis=1) # only take the max prob value
    for i in range(min(limit, len(prob))):
        ind = np.argmax(max_prob)
        if max_prob[ind] > thres:
            y_test[ind] = res[ind]
            max_prob[ind] = 0.0 # replace the prob
    clusters[clusters == -1] = y_test
    return clusters
# Phase 1: Clustering scripts using DBSCAN
dbscan = DBSCAN(eps=0.1, min_samples=3, metric='dice', algorithm='auto')
labels = dbscan.fit(script_features).labels_
# Phase 2: Merge clusters
first_max = None, last_max = None
while True:
    res = pairwise_cluster_comparison(script_features, labels)
    last_max = max(res.values())
    maxs = [i for i, j in res.items() if j == last_max]
    first_max = last_max if first_max == None
    if first_max - last_max < 0.05:
        largest_cluster, selected = 0, None
        for x,y in maxs:
            f1 = len(np.where(labels == x)[0])
            f2 = len(np.where(labels == y)[0])
            if largest_cluster < max(f1, f2):
                selected = (x, y) if f1 > f2 else (y, x)
                largest_cluster = max(f1, f2)
        labels[labels == selected[1]] = selected[0]
    else: break
# Phase 3: Classify noisy samples
final_label = None
while True:
    nlabels = classification(script_features, labels, 0.7, 5)
    if np.array_equal(nlabels, labels):
        final_label = labels
        break
    else: labels = nlabels
```

Listing 1: Code for different phases of clustering scripts.

B EXAMPLE SENSOR-ACCESSING SCRIPTS

```
dvObj.pubSub.subscribe(rtnName, impId, 'SenseTag_RTN', function() {
    try {
        var maxTimesToSend = 2;
        var avgX = 0, avgY = 0, avgZ = 0, avgX2 = 0, avgY2 = 0, avgZ2 = 0, countAcc = 0, accInterval = 0;
        function dvDoMotion() {
            try {
                if (maxTimesToSend <= 0) {
                    window.removeEventListener('devicemotion', dvDoMotion, false);
                    return;
                }
                var motionData = event.accelerationIncludingGravity;
                if ((motionData.x) || (motionData.y) || (motionData.z)) {
                    var isError = 0; var x = 0; var y = 0; var z = 0;
                    if (motionData.x) x = motionData.x;
                    else isError += 1;
                    if (motionData.y) y = motionData.y;
                    else isError += 1;
                    if (motionData.z) z = motionData.z;
                    else isError += 1;
                    avgX = ((avgX * countAcc) + x) / (countAcc + 1);
                    avgX2 = ((avgX2 * countAcc) + (x * x)) / (countAcc + 1);
                    avgY = ((avgY * countAcc) + y) / (countAcc + 1);
                    avgY2 = ((avgY2 * countAcc) + (y * y)) / (countAcc + 1);
                    avgZ = ((avgZ * countAcc) + z) / (countAcc + 1);
                    avgZ2 = ((avgZ2 * countAcc) + (z * z)) / (countAcc + 1);
                    countAcc++;
                    accInterval = event.interval;
                    if (countAcc % 400 == 1) {
                        maxTimesToSend--;
                        sensorObj = {};
                        sensorObj['MED_AMtX'] = Math.max(Math.min(avgX, 10000), -10000).toFixed(7);
                        sensorObj['MED_AMtY'] = Math.max(Math.min(avgY, 10000), -10000).toFixed(7);
                        sensorObj['MED_AMtZ'] = Math.max(Math.min(avgZ, 10000), -10000).toFixed(7);
                        sensorObj['MED_AVrX'] = Math.max(Math.min((avgX2 - avgX * avgX), 10000), -10000).toFixed(7);
                        sensorObj['MED_AVrY'] = Math.max(Math.min((avgY2 - avgY * avgY), 10000), -10000).toFixed(7);
                        sensorObj['MED_AVrZ'] = Math.max(Math.min((avgZ2 - avgZ * avgZ), 10000), -10000).toFixed(7);
                        sensorObj['MED_ANum'] = countAcc;
                        sensorObj['MED_AInterval'] = accInterval;
                        dvObj.registerEventCall(impId, sensorObj, 2000, true);
                    }
                }
            } catch (e) {}
        }
        setTimeout(function() {
            try {
                if (window.addEventListener == undefined) return;
                window.addEventListener('devicemotion', dvDoMotion);
            } catch (e) {}; }, 3000);
    } catch (e) {};
});
```

Listing 2: JavaScript snippet from `doubleverify.com` computing average and variance of accelerometer data.

```
"https://tps10212.doubleverify.com/event.gif?impid=0b86b20d52a84923a41d85da169fd97f&msrdp=1&naral=80&vct=1&engalms=83&engisel=1&
    MED_AMtX=2.8139038&MED_AMtY=7.6222534&MED_AMtZ=4.0931549&MED_AVrX=0.0000000&MED_AVrY=0.0000000&MED_AVrZ=0.000
    0000&MED_ANum=1&MED_AInterval=16.666&cbust=1508977547663635"
```

Listing 3: `doubleverify.com` script sending average and variance of sensor data as URL parameters to their servers.

```
# collecting sensor data
cdma : function (t) {
            try {
                if (cf[_ac[109]] < cf[_ac[254]] && cf[_ac[497]] < 2 && t) {
                    var e = cf[_ac[374]]() - cf[_ac[253]], c = -1, n = -1, a = -1;
                    t[_ac[157]] && (
                        c = cf[_ac[554]](t[_ac[157]][_ac[413]]),
                        n = cf[_ac[554]](t[_ac[157]][_ac[190]]),
                        a = cf[_ac[554]](t[_ac[157]][_ac[524]]) );
                    var o = -1, f = -1, i = -1;
                    t[_ac[310]] && (
                        o = cf[_ac[554]](t[_ac[310]][_ac[413]]),
                        f = cf[_ac[554]](t[_ac[310]][_ac[190]]),
                        i = cf[_ac[554]](t[_ac[310]][_ac[524]]) );
                    var r = -1, d = -1, s = 1;
                    t[_ac[526]] && (
                        r = cf[_ac[554]](t[_ac[526]][_ac[62]]),
                        d = cf[_ac[554]](t[_ac[526]][_ac[548]]),
                        s = cf[_ac[554]](t[_ac[526]][_ac[515]]) );
                    var u = cf[_ac[109]] + _ac[66] + e + _ac[66] + c + _ac[66] + n + _ac[66] + a +
                            _ac[66] + o + _ac[66] + f + _ac[66] + i + _ac[66] + r + _ac[66] + d + _ac[66] +
                            s + _ac[379];
                    cf[_ac[496]] = cf[_ac[496]] + u,
                    cf[_ac[68]] += e,
                    cf[_ac[335]] = cf[_ac[335]] + cf[_ac[109]] + e,
                    cf[_ac[109]]++
                }
                cf[_ac[69]] && cf[_ac[109]] > 1 && cf[_ac[419]] < cf[_ac[626]] && (
                    cf[_ac[120]] = 7,
                    cf[_ac[609]](),
                    cf[_ac[194]](!0),
                    cf[_ac[340]] = 1,
                    cf[_ac[419]]++ ),
                cf[_ac[497]]++
            } catch (t) {}
        },
# sending encoded data to remote server
apicall_bm: function (t, e, c) {
            var n;
            void 0 !== window[_ac[175]]
                ? n = new XMLHttpRequest
                : void 0 !== window[_ac[637]]
                    ? (n = new XDomainRequest, n[_ac[158]] = function () {
                        this[_ac[269]] = 4,
                        this[_ac[567]] instanceof Function && this[_ac[567]]()})
                    : n = new ActiveXObject(_ac[450]),
            n[_ac[587]](_ac[291], t, e),
            void 0 !== n[_ac[360]] && (n[_ac[360]] = !0);
            var a = cf[_ac[258]](cf[_ac[75]] + _ac[85]);
            cf[_ac[302]] = _ac[123] + a + _ac[611],
            void 0 !== n[_ac[279]] && (
                n[_ac[279]](_ac[534], _ac[115]),
                cf[_ac[302]] = _ac[538]);
            var o = _ac[266] + cf[_ac[261]] + _ac[611] + cf[_ac[302]] + _ac[100];
            n[_ac[567]] = function () {
                n[_ac[269]] > 3 && c && c(n)
            },
            n[_ac[101]](o)
        },
```

Listing 4: Obfuscated JavaScript snippet from `homedepot.com/_bm/async.js` collecting sensor data

C SENSOR DATA EXFILTRATION: EXAMPLE PAYLOADS

```
"parameter": url$0$https://mobile.reuters.com/"," referrer$ 0$" , " ancestorOrigins$ 0$n/a" , "video$0$360x592x24", "frame$0$0", "hidden$0$0", "
    visibilityState$ 1 $visible " , "window$1$344x521","inner$1$360x521","outer$1$360x592", " localStorage$ 3$1" , " sessionStorage$ 3$1" , "
    appCodeName$4$Mozilla","appName$4$Netscape","appVersion$4$5.0 (Android 7.0)", "cookieEnabled$4$true" , "doNotTrack$4$unspecified", "
    hardwareConcurrency$4$8","language$5$en−US","platform$5$Linux armv7l","product$5$Gecko","productSub$5$20100101","sendBeacon$5$1",
    "userAgent$5$Mozilla/5.0 (Android 7.0; Mobile; rv:55.0) Gecko/55.0 Firefox/55.0" , "vendor$5$", "vendorSub$5$","fontrender$8$1" , "
    webgl$239$1","time$240$1526528127627","timezone$240$0", "plugins$240$None","time− fetchStart$ 241$321" , "time−domainLookupStart$241
    $324","time−domainLookupEnd$241$324","time−connectStart$241$324","time−connectEnd$241$339","time−requestStart$241$367","time−
    responseStart$241$383","time−responseEnd$241$414","time−domLoading$241$418","time−domInteractive$241$4533","time−
    domContentLoadedEventStart$241$4828","time−domContentLoadedEventEnd$241$4966","navigation−redirectCount$241$0","navigation−
    type$241$navigate","globals−time$266$0.705","globals$269$a8bb2f85","document−time$272$0.43","document$274$0a886779","clock$288$662","
     intersection$ 293$n/a" , " battery$ 299$1 1 0 Infinity " , " devicelight$ 325$987" , "framerate$461$10" , " sort$ 763$121.685" , " deviceproximity$
    817$3" , "userproximity$1295$near" , " orientation$ 1297$43.123402478330654 32.98760746072672 21.654300663242278" , "motion$1305$
    0.12560407138550994 -0.12339847737456243 -0.18449644504208473" , "audiocontext$2044$d554bfaa
```

Listing 5: Orientation, motion, light and proximity data is sent to https://api-34-216-170-51.b2c.com/api/x?parameter=[...] on `reuters.com` website

```
"{"{ \"is_supposed_final_message\" : false , \"message_number\":1,\"message_time\":7736, \" query_string \" : \"e" : "36" , "ue" : "1" , "uu" : "1" , "qa" : "
    360" , "qb" : "592" , "qc" : "0" , "qd" : "0" , "qf" : "360" , "qe" : "521" , "qh" : "360" , "qg" : "592" , "qi" : "360" , "qj" : "592" , "ql" : "" , "qo" : "0" , "qm":"0" , ...
    , \"user_agent\" : \" Mozilla/5.0 (Android 7.0; Mobile; rv:55.0) Gecko/55.0 Firefox/55.0\" , ... , \" location \" : \" https://i. stuff .co.nz
    /\" , \" referrer \" : \" https://www.stuff.co.nz/\" , ... , \"numbers\":[−9986.46115497749,9007199254740994,6.283185307179586,5.43656365691
    809,0.9150885074842403,−1.8369701987210297e−16,−0.5401928810015185,5.551115123125783e−16,7.600875484570224], \" plugins \" : \" 101574
    \" , \" graphics_card \" : {} , \" cpu_cores\" : 8 , \" canvas_render \" : 1490412693, \" webgl_render\" : 3707405031, \" installed_fonts \" : [ \" Dotum\",
    \"DotumChe\",\"Gulim\", \" GulimChe\",\" Malgun Gothic\" , \" Meiryo UI\" , \" Microsoft JhengHei\" , \" Microsoft YaHei\" , \" MONO\",\"MS UI
    Gothic\"], ... , \"RTCPeerConnection\":\"RTCPeerConnection\",\"WebSocket\": \" WebSocket\", \" unloadEventStart \" : 0, \" unloadEventEnd\":0,
    \" sahimap\": \" TypeError: window.SahiHashMap is undefined\"}, \" color_depth \" : 24, \" min_safe_int \" : −9007199254740991,\"gz\" : false , \"
    gz_cde\" : false , \" input \" : { \" key_times\" : [] , \" key_delta_mean\":−1, \" key_delta_var \" : −1, \" mouse\":[] , \" mousedowns\":[], \"
     orientation \" : [[1526540170271,[ false ,43.123406989295376,32.98760380966044,21.654305428432068]] , [1526540175119,[ false ,
    43.123405666163535,32.987604652675195,21.654303695580264]]] , ... }"}"
```

Listing 6: Orientation data is sent to https://px2.moatads.com/pixel.gif?v=[...] on `stuff.co.nz` website

```
{payload=[{"t" : "PX164","d" : {"PX165":["0.12560246652278528, -0.12339765619546963, -0.18449742637532154", "0.12560876871457533,
              -0.12339147047919652, -0.18449095921522524", "0.1256049922328881, -0.12339788204758717, -0.1844980715878096", "
    0.12560647137074932, -0.12339009836285393, -0.1844992891051791"] , "PX63":"Linux armv7l", "PX371":true}}]&appId=PXpHWOqUmu&tag
    =v3.19.1&uuid=b3b264b0−5987−11e8−8ef7−216942b9c24f&ft=24&seq=3&cs=a829d69e16358d1daa46d1266c00266e44d900e411d6666bb1b4d9d9
    08e1827c&pc=7889002194399290&sid=b3b8f462−5987−11e8−ae82−6db5f4392e69&vid=b3b8f460−5987−11e8−ae82−6db5f4392e69}
```

Listing 7: Motion data are sent to https://www.kayak.com/px/xhr/api/v1/collector in base64-encoded form (decoded here) on `kayak.com` website