A Large-scale Study on the Risks of the HTML5 WebAPI for Mobile Sensor-based Attacks

Francesco Marcantoni  
Univ. of Illinois at Chicago, USA  
fmarca2@uic.edu

Sotiris Ioannidis  
FORTH, Greece  
sotiris@ics.forth.gr

Michalis Diamantaris  
FORTH, Greece  
diamant@ics.forth.gr

Jason Polakis  
Univ. of Illinois at Chicago, USA  
polakis@uic.edu

ABSTRACT

Smartphone sensors can be leveraged by malicious apps for a plethora of different attacks, which can also be deployed by malicious websites through the HTML5 WebAPI. In this paper we provide a comprehensive evaluation of the multifaceted threat that mobile web browsing poses to users, by conducting a large-scale study of mobile-specific HTML5 WebAPI calls used in the wild. We build a novel testing infrastructure consisting of actual smartphones on top of a dynamic Android app analysis framework, allowing us to conduct an end-to-end exploration. Our study reveals the extent to which websites are actively leveraging the WebAPI for collecting sensor data, with 2.89% of websites accessing at least one mobile sensor. To provide a comprehensive assessment of the potential risks of this emerging practice, we create a taxonomy of sensor-based attacks from prior studies, and present an in-depth analysis by framing our collected data within that taxonomy. We find that 1.63% of websites could carry out at least one of those attacks. Our findings emphasize the need for a standardized policy across browsers and the ability for users to control what sensor data each website can access.

CCS CONCEPTS

• Security and privacy → Mobile platform security.

KEYWORDS

Android; mobile HTML5; sensor attack taxonomy; WebAPI

ACM Reference Format:

1 INTRODUCTION

Smartphones have become almost ubiquitous, with the volume of Internet traffic from mobile devices surpassing that of desktop computers worldwide [35], while 56% of the traffic to top-sites in the US was from mobile devices [65]. Apart from the obvious usability benefits smartphones offer, they have also introduced a plethora of risks. Users are becoming increasingly aware of privacy issues including online tracking and internet surveillance, and employ private browsing among other techniques to remain anonymous online [70, 71]. However, adversaries can still track users through fingerprints [53], making it possible to identify which device is navigating a given webpage [42]. Prior work has also shown that imperfections in sensors’ hardware render them fingerprintable [25]. Websites can access such mobile sensor data through the widely supported HTML5 WebAPI. Thus, a plethora of attacks that were previously limited to mobile apps can “migrate” to the mobile web, as modern browsers provide access to a device’s sensors.

In this paper we present a quantitative and qualitative large-scale study of mobile-specific WebAPI calls made by websites in the wild. We build a unique crawling infrastructure that uses Android devices and perform an end-to-end analysis of WebAPI requests. Using our crawling infrastructure, we measure the prevalence of mobile-specific WebAPI calls across 183,571 of the most popular websites during March-November 2018. Our experiments capture the true scale of this phenomenon, as we detect 5,313 unique domains accessing at least one mobile WebAPI call; 35.89% of those also result in sensors being accessed by third-party scripts. To better understand the implications we survey prior literature on attacks from malicious apps that leverage data from mobile sensors. Based on this diverse yet representative selection of papers we create a taxonomy of attacks that could be potentially carried out by modern websites and conduct an in-depth analysis of our dataset. We argue that with browsers enforcing different access policies, as do the plethora of apps that support WebView, there is dire need for a standardized, fine-grained universal mechanism that allows users to control access to all types of mobile sensor data.

Overall, this paper makes the following contributions:

• We build a novel crawling infrastructure and conduct a high-fidelity, large-scale end-to-end study of websites targeting mobile-specific sensors.

• We provide a taxonomy of previously reported sensor-based attacks and reframe them within the modern mobile ecosystem. Guided by our taxonomy we conduct a qualitative analysis of our collected data and provide a comprehensive assessment of the threat posed by the mobile WebAPI.

• We publicly release our data: https://www.cs.uic.edu/~webapi.
2 BACKGROUND

Attack Taxonomy. A plethora of research papers have demonstrated mobile-based attacks that employ sensor data. While a considerable number of attacks present similar characteristics, e.g., demonstrating different techniques for inferring a user’s touchscreen input or fingerprinting the user’s device, a wide range of different attacks have been proposed. Here we introduce a taxonomy of attacks compiled from the literature that captures the vast potential of how mobile sensor data can be misused by adversaries. Typically these attacks assume that attackers are able to obtain sensor data through the HTML5 WebAPI. This leads to changes in the position of the screen, its orientation and the touchscreen (e.g., [46, 56]). This is possible because typing leads to changes in the position of the screen, its orientation and the device’s motion. In a different study, the Light sensor was used to identify the content of an external display and even classify users’ digital activities into different categories with an 85% accuracy [17].

User tracking. User tracking has garnered much attention [9, 10, 15, 20–23, 25, 30, 36, 39, 40, 47, 52, 58, 59, 73, 74], and can be conducted in different ways – from coarse-grained location tracking that does not require any user-permission (using the Accelerometer or Gyroscope) [36, 52], to fine-grained device fingerprinting using rich and high-resolution data from smartphone sensors (e.g., [10, 25]). In this category we also include alternative attacks that could track users by inferring demographic information (e.g., age [23], gender [47] and fingerprints [30]), physical traits such as their gait [40, 59], or information about their mental state or mood [74].

Table 1: Sensor based side-channel attacks.

<table>
<thead>
<tr>
<th>Ref. #</th>
<th>Attack</th>
<th>Motion</th>
<th>Orientation</th>
<th>Light</th>
<th>Physical</th>
<th>GPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[36, 38]</td>
<td>Mode of transportation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[52]</td>
<td>Movement/activity</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[58]</td>
<td>Location Tracking</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[45]</td>
<td>Acoustic emanation side-channels</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[44]</td>
<td>Speech recognition</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[33, 35, 46, 62]</td>
<td>Touchscreen input</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[56]</td>
<td>OS/app fingerprinting</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[44]</td>
<td>Physical or demographics</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[41, 22, 29]</td>
<td>Device/sensor fingerprinting</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>[57]</td>
<td>Covert communication side-channels</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Sensors marked with (●) are sufficient for performing the specific attack. When a combination of multiple sensors is required to perform the attack, they are marked with (●). We denote optional sensors that enhances the accuracy of the attack with (●). Grey columns denote sensor data that should require explicit user permission according to the W3C.
Deconstructing sensor attacks. Table 1 lists the attacks described in the studies that guided our taxonomy. We classify previous attack papers based on the type of sensor data needed to carry out the attacks. For example [58] infers the body movement or activity of a user by accessing the motion and GPS sensors. On the other hand, [52] only requires the orientation sensor, but using the Motion or Magnetometer sensor can further improve the attack.

3 METHODOLOGY AND SYSTEM DESIGN

In this section we present our system design and experimental methodology. We give an overview of our system’s architecture, and provide implementation details about the in-line hooking methods.

System architecture. Our system employs a proxy server that intercepts network traffic by using mi tmpoxy [18]. We configured all the Android devices used in our experiments with mi tmpoxy’s certificate in order to intercept both HTTP and HTTPS traffic. The proxy server injects a JavaScript component that hooks and monitors JavaScript calls to mobile-specific WebAPIs. However, our aim is to obtain an in-depth view of sensor data access. In general, browsers are responsible for mediating access between high-level JavaScript function calls and low-level Android API calls. Understanding how this mechanism works for every browser would be time consuming and in many cases infeasible due to proprietary code. As such, we have opted for a generic and browser-agnostic approach, where we intercept Android system calls using a custom Xposed [60] module that (i) detects and hooks requests to sensor-specific Android API calls and (ii) identifies which of these API calls are permission-protected. By intercepting these low-level function calls we are able to validate that the JavaScript interception was successful and calls requesting sensor data were correctly logged.

Mobile HTML5 Functions. We identified the functions that retrieve mobile-specific data through the official mobile HTML5 WebAPI [31]. We consider as mobile-specific any calls that obtain information originating from an integrated sensor of a mobile device. The HTML5 WebAPI calls interact with the webpage using either a JavaScript calls to mobile-specific WebAPIs. However, our aim is to obtain an in-depth view of sensor data access. In general, browsers are responsible for mediating access between high-level JavaScript function calls and low-level Android API calls. Understanding how this mechanism works for every browser would be time consuming and in many cases infeasible due to proprietary code. As such, we have opted for a generic and browser-agnostic approach, where we intercept Android system calls using a custom Xposed [60] module that (i) detects and hooks requests to sensor-specific Android API calls and (ii) identifies which of these API calls are permission-protected. By intercepting these low-level function calls we are able to validate that the JavaScript interception was successful and calls requesting sensor data were correctly logged.

Mobile HTML5 Functions. We identified the functions that retrieve mobile-specific data through the official mobile HTML5 WebAPI [31]. We consider as mobile-specific any calls that obtain information originating from an integrated sensor of a mobile device. The HTML5 WebAPI calls interact with the webpage using either a JavaScript calls to mobile-specific WebAPIs. However, our aim is to obtain an in-depth view of sensor data access. In general, browsers are responsible for mediating access between high-level JavaScript function calls and low-level Android API calls. Understanding how this mechanism works for every browser would be time consuming and in many cases infeasible due to proprietary code. As such, we have opted for a generic and browser-agnostic approach, where we intercept Android system calls using a custom Xposed [60] module that (i) detects and hooks requests to sensor-specific Android API calls and (ii) identifies which of these API calls are permission-protected. By intercepting these low-level function calls we are able to validate that the JavaScript interception was successful and calls requesting sensor data were correctly logged.

Mobile HTML5 Functions. We identified the functions that retrieve mobile-specific data through the official mobile HTML5 WebAPI [31]. We consider as mobile-specific any calls that obtain information originating from an integrated sensor of a mobile device. The HTML5 WebAPI calls interact with the webpage using either a JavaScript calls to mobile-specific WebAPIs. However, our aim is to obtain an in-depth view of sensor data access. In general, browsers are responsible for mediating access between high-level JavaScript function calls and low-level Android API calls. Understanding how this mechanism works for every browser would be time consuming and in many cases infeasible due to proprietary code. As such, we have opted for a generic and browser-agnostic approach, where we intercept Android system calls using a custom Xposed [60] module that (i) detects and hooks requests to sensor-specific Android API calls and (ii) identifies which of these API calls are permission-protected. By intercepting these low-level function calls we are able to validate that the JavaScript interception was successful and calls requesting sensor data were correctly logged.

JavaScript Calls Interception. We build our component for hooking JavaScript methods upon the javascript-hooker Node.js module [13]. In our experiments we do not overwrite the original function but only need to identify whether a function is called. Thus, whenever a WebAPI is called the JavaScript modules creates a log entry for further analysis and executes the original function. Since javascript-hooker also takes the arguments of the original function, we can also intercept the arguments of the addEventListener and check for events of interest. Our code is injected in the head of the document (if there is one) or the page body otherwise.

In order to listen to events and associate a function to a specific target we need to intercept the setter property. Even though this is possible using Object.defineProperty() the original value will be lost and the webpage may not function as expected. Therefore, we follow the approach employed by Chameleon [34] and overwrite the getter property of each event prototype. As such, every time the property is read, our custom function is called. While certain sensor data may normally remain the same during navigation (e.g., properties related to display characteristics), remaining constant might be considered “suspicious” for other sensors. For instance, when an actual human uses the device, small changes in the gyroscope readings would be expected. As such, to make our crawling more realistic, our system intercepts the values returned by certain sensors and slightly modifies their value. In general, data retrieved through events, is handled in two different ways: listening to addEventListener on the target object while checking if the argument matches the desired event and defining new getters for the properties of the event’s prototype.

Identifying the JavaScript source. Apart from logging WebAPI calls we also want to identify the origin of the JavaScript files being executed. This information is important in order to identify if the script belongs to a first-party domain or a third-party domain. We register the source of the URL by utilizing the stack property of the Error object. Our hooking script implements a mechanism that creates an Error object and reads its stack property.

Android API call interception. Each mobile HTML5 WebAPI is associated with a low-level Android API call. In order to validate the results of the JavaScript interception and to identify which ones require a permission, we use the PermissionHarvester [26] module that hooks every Android permission protected API call. Since access to some of the sensors does not require an Android permission, we also manually identified and hooked the functions that give access to non-permission-protected sensor data. Android apps (including the browser) cannot directly read the current value of a sensor and are required to register a listener in order to consequently read the captured events. Each sensor can be obtained by declaring a listener and specifying its name with the getDefaulSensor() function. Then, the listener is registered using the registerListener() method. Our module intercepts both of these function calls.

Experimental setup. In our experiments we use Mozilla Firefox (v.59.0.1) as our browser on three Android Google Nexus 5X and a OnePlus One device, all running AOSP 7.1.2. We controlled the devices using the Android Debug Bridge. We first evaluated the effectiveness of our methodology by creating a dummy website that executes all possible mobile HTML5 WebAPIs. Since browsers require a valid certificate in order to call certain APIs we used a self-signed certificate and confirmed that our approach can successfully intercept and monitor access to the devices’ sensors. Our system also simulates brief user interaction through random gestures which are issued for approximately 30 seconds on average for each website, while an extra module monitors for potential redirections to different domains which are rolled back so the original website can continue to be processed.

4 DATA COLLECTION AND ANALYSIS

Our crawling list included the 200K most popular websites according to Alexa, as returned on 03/24/2018. Our system was unable to access or complete the crawling process for 16,199 (8%) of the domains in our list (e.g., 503-timeout, 502-Bad Gateway, or DNS errors), and omitted 230 domains flagged as malicious by the Google

In Table 2 we can see the prevalence of the mobile-specific WebAPI calls logged by our system among the 183,571 domains processed by our crawling infrastructure. We logged 5,313 (2.89%) websites using at least one of the targeted APIs, while 807 request access to sensor data using more than one of the API calls. The most prevalently accessed data is from the acceleration and orientation sensors which do not require the user’s permission, as well as geolocation data which requires permission in major browsers. While the Geolocation API can also return information for desktop computers (using “information about nearby wireless access points and the IP address” [4]), we consider it mobile-specific due to smartphones’ integrated GPS receivers which provide real-time location information. While geolocating users based on landline IP addresses is considerably accurate [68], that is not the case for mobile IP addresses [12, 66]. It is important to note that the Media capture and Geolocation APIs should explicitly request permissions from the user; while this is enforced in major browsers, it is not always the case with other browsers (e.g., for Geolocation [41]). For the remaining WebAPI calls, users will be unaware that such information is being retrieved by the website even for major browsers.

As Figure 2 shows the use of mobile-specific WebAPI calls is not uniform across our dataset. The highest concentration is found in the top 5K websites with 250 domains. We observe that domains also access more permission-free WebAPIs (gray bars) independently of their rank, indicating the importance of this sensor data. This type of information can be used for a plethora of attacks, and users should have the ability to explicitly grant permission for them.

As Figure 3 (left) shows the majority of websites issue request access to a single sensor through the WebAPI, while 15.1% of the domains we processed target at least two different types of sensor data. As shown in Table 1, only accessing the Motion sensor can lead to six different attacks, while a combination of two sensors (Motion and Orientation) leads to eight attacks. Furthermore, as can be seen in Figure 3 (right) 56.6% of the domains that issue mobile-specific WebAPI calls are able to perform at least one attack.

### Sensor-based attacks

We continue our analysis by framing our dataset within our taxonomy based on representative prior work. It is important to note that in our analysis we do not take into account or argue for (or against) the plausibility of the attacks presented in previous studies. Instead, our goal is to measure the potential risk that mobile users face due to web browsing by identifying websites that request access to specific sensor data and could potentially misuse them in an invasive or malicious manner.

Table 3 breaks down the number of domains for each attack. We observe that the most common attacks across websites that access WebAPIs are device/sensor fingerprinting and trait, mood or demographic inference (54.07%), location tracking and mode of transportation (53.84%), speech recognition (41.38%) and touchscreen input (26.82%). We argue that any information gained from sensors poses a risk for users and an access control policy should be enforced, either through some form of run-time permissions [5] or using a mechanism similar to GDPR [3].

### Banking sites

While fingerprinting allows third parties to track users across the Web [53], fingerprints can be used as an additional factor for authentication [8]. As such, banking websites are well-suited for deploying such a security mechanism [54] due to the significant implications of compromised accounts. As details of such practices are not typically disclosed, we further explore the prevalence of sensor-based information access across e-banking domains. We compiled a list of bank domains using [1, 6] and cross-referenced it with our dataset. We identified 65 banking domains that request access to at least one mobile sensor. Banking domains
request access for 1.38 sensors on average, which is higher than the average of 1.17 in other domains, indicating that they are more likely to leverage the HTML5 WebAPI for accessing sensor data. We find that 24 of the domains obtain access to the sensor data necessary to conduct at least one of the attacks included in our taxonomy. Interestingly, all of those banks collect the sensor data leveraged in prior work for device fingerprinting, while 40 banks request access to the user’s geolocation which can also be used for enhancing the authentication process [8]. We find that efirstbank.com actually requests access to more sensors than any other domain in our entire dataset. Overall, while accessing sensor data could be motivated by enhancing the authentication process, this practice raises privacy concerns as argued by privacy advocates [48].

Request origin. Next we explore the origin of WebAPI requests (first or third party) and whether it was included in an iframe. Different browsers implement different policies regarding which sensors can be accessed for these three different types of origin. We present statistics for all the websites that requested access, even if those requests were blocked by Firefox during our experiments.

Iframes. Our system collects all the calls executed by every element of a website, including iframes. In every log we record the source domain name of the element that is accessing sensor information. By comparing the URL of the address bar and the URL in the logfiles, we can identify whether WebAPIs are accessed by the DOM or by an iframe. Our analysis shows that 991 websites out of 5,313 contain iframes that use WebAPIs to access mobile specific information. We analyzed all iframes from our experiments and found that specific iframes are found in different websites. The two most frequent domains injected inside iframes exist in 389 webpages (or 39.3% of pages with iframes collecting data) and are related to online media players.

External sources. Among the websites that issue API calls for mobile specific information we found 40 scripts from external domains (either as a third-party scripts or inside an iframe) that collect data from 2461 websites 46.3%. We manually analyzed these scripts and found that they offer services for media-players and advertisements and they collect information about the orientation and motion of the device. In Table 4 we list the domains that appear in more than 50 websites and collect data from sensors. The first column is the origin of the script being executed. The second and third column show how many websites and iframes host this script. Given that these third-party domains are used in 35.89% of websites that access sensor data, we classified them based on the type of service they provide using Cyren. The last two columns show which sensors the script accessed and their corresponding attacks. We observe that most of these domains call the motion and orientation WebAPIs which enable a plethora of attacks. Moreover, domains classified as search engines and ad-networks gain access to characteristics that can track users across the web.

From Table 4 we can see that the domain api.b2c.com enables 12 different attacks. After investigating this domain through VirusTotal [7] we found that scripts served from this domain and Android apps that communicate with it are classified as intrusive adware and even malware by some antivirus vendors. Another domain, c.adsco.re, is flagged as malware by Cyren, even though it is not considered malicious by the Google SafeBrowsing API. We manually analyzed the content of the script that retrieves the data and found that apart from retrieving information about the Motion and the Orientation sensors it also exhibits behavior which is a strong indicator of device fingerprinting, such as creating and manipulating canvas elements [50] and reading different Navigator, Screen, Storage and Window properties. Interestingly the adsco.re domain states that it is used for traffic validation by Adscore, a bot detection service. In total, these two domains which are considered malicious by certain security lists, were found on 5.4% of all the sensor-accessing domains logged by our system, which again raises concerns regarding browser policies that allow third party domains to access sensor data without explicit user permission.

Android internals. Our crawling system allows an end-to-end analysis of sensor data access. Apart from providing high call-detection fidelity, since we can match requests logged by our injected JavaScript to actions at the operating system level, it also revealed sub-optimal browser behavior. We found that while Firefox prevents iframes from accessing sensor data, in practice Firefox simply "omits" returning the sensor data instead of blocking (i.e., ignoring) the actual request. Specifically, Firefox allows iframes to create event listeners, which then trigger the necessary WebAPI calls which then trigger the corresponding Android-level processing and permission checks for obtaining the sensor data; the data is then returned to the browser but not provided to the iframe.

Malicious domains. Even though our system checked Google’s SafeBrowsing API before visiting a domain, it is possible that visited domains could be flagged as malicious later on, or by different blacklists. As such, we submitted all the domains that issued WebAPI requests to VirusTotal. Figure 4 presents the websites flagged as malicious (sorted by their rank), the number of accessed sensors per website and feasible attacks. Out of those, 149 domains were flagged by one AV engine and 17 domains were flagged by two. We can see that higher ranking malicious domains are more likely to access more sensors which results in a higher number of feasible attacks. We found 11 websites being flagged by at least 3 AV engines. Finally, we found two websites, namely google.com and youtube.com, that are flagged by eight AV engines as malicious. Apart from likely examples of typosquatting [49, 69], these websites...
Table 4: Third-party scripts accessing WebAPI calls.

<table>
<thead>
<tr>
<th>Script origin</th>
<th>Sites</th>
<th>iFrames</th>
<th>Sensors</th>
<th>AttackID</th>
</tr>
</thead>
<tbody>
<tr>
<td>foxnews-cco.com</td>
<td>275</td>
<td>0</td>
<td>0</td>
<td>1, 3, 5, 9</td>
</tr>
<tr>
<td>fast.wisita.com</td>
<td>467</td>
<td>3</td>
<td>QC</td>
<td></td>
</tr>
<tr>
<td>fast.wisita.net</td>
<td>123</td>
<td>115</td>
<td>QC</td>
<td></td>
</tr>
<tr>
<td>c.alicdn.com</td>
<td>211</td>
<td>0</td>
<td>M, O, P, L</td>
<td>1-6, 8, 9, 11-13</td>
</tr>
<tr>
<td>g.alicdn.com</td>
<td>170</td>
<td>65</td>
<td>QG</td>
<td>1, 3, 5, 9</td>
</tr>
<tr>
<td>aeu.alicdn.com</td>
<td>127</td>
<td>83</td>
<td>O</td>
<td>1, 3, 5, 9</td>
</tr>
<tr>
<td>api.bzc.com</td>
<td>76</td>
<td>0</td>
<td>M, O, P, L</td>
<td>1-9, 11-13</td>
</tr>
<tr>
<td>ccdnadmizxer.net</td>
<td>169</td>
<td>-</td>
<td>M</td>
<td>1, 3, 4, 6, 8, 11-13</td>
</tr>
<tr>
<td>static.yieldmo.com</td>
<td>107</td>
<td>-</td>
<td>0</td>
<td>1, 3, 5, 9</td>
</tr>
<tr>
<td>secure.data.serving-sys.com</td>
<td>51</td>
<td>31</td>
<td>M</td>
<td>1, 3, 4, 6, 8, 11-13</td>
</tr>
<tr>
<td>dlwbr badau.com</td>
<td>77</td>
<td>69</td>
<td>M</td>
<td>1, 3, 4, 6, 8, 11-13</td>
</tr>
<tr>
<td>client.permeters.net</td>
<td>73</td>
<td>-</td>
<td>M</td>
<td>1, 3, 4, 6, 8, 11-13</td>
</tr>
</tbody>
</table>


5 RELATED WORK

The WebAPI has standardized many features providing greater support for developers and improving the user experience [57]. Snyder et al. [63] presented a cost-benefit analysis of the WebAPI using a small set of websites (10K) and focusing on desktop browsing.

In an independent and concurrent recent study Das et al. [19] presented a study on web scripts accessing mobile sensors. While their study also targets WebAPI calls for mobile sensors, our work presents significant differences. In regards to the actual datasets, our study is on a considerably larger set of domains while also having little overlap due to the fluctuation of the Alexa list [61]. Moreover, their system detects a subset of the mobile-specific WebAPI calls handled by our system, and their study focuses on sensor-based fingerprinting thus offering a limited examination of the risks that users face; we frame our findings within our attack taxonomy and provide a more comprehensive evaluation of the feasibility of a wide range of sensor-based attacks. Furthermore, our crawling infrastructure uses actual mobile devices and provides a unique end-to-end view of data requests and access, while their crawlers rely on a modified version of OpenWPM running on desktop machines which could be detected by evasive websites [43].

Browser fingerprinting has gathered a lot of attention and the research community has extensively studied the techniques that make it possible [28, 29]. With the growing usage of smartphones, traditional desktop fingerprinting techniques [67] are becoming less effective as some information is being standardized in many mobile browsers [39]. On the other hand, the development of new mobile-specific HTML5 WebAPIs offered new avenues for trackers to exploit other types of data that were not present in desktops. As previous work [9, 10, 15, 20–23, 25, 30, 36, 39, 40, 47, 52, 55, 58, 59, 73, 74] has shown, the huge amount of input collected by smartphones sensors resulted in new opportunities for device fingerprinting.

6 CONCLUSION

We presented a comprehensive evaluation of the threats that mobile users face when browsing the Web, due to capabilities offered by modern browsers, by conducting the largest and most extensive study to date on the use of mobile-specific WebAPI calls in the wild. Our study was conducted using a novel crawling infrastructure built on top of actual smartphones. Our findings demonstrate that WebAPI capabilities are actively being used by websites for accessing mobile sensors. To provide the appropriate context that highlights the true threat posed by this practice, we created a taxonomy of sensor-based attacks compiled from a wide range of attacks demonstrated in prior work. Our subsequent in-depth analysis correlated the sensor data currently being accessed by websites and the data-requirements of prior attacks, leading to several alarming findings. We argue that our findings support the need for more stringent policies for websites attempting to access sensor data.

ACKNOWLEDGMENTS

This project has received funding from Horizon 2020 under grant agreements No 786890 and No 740787, the DARPA ASED Program and AFRL under contract FA8650-18-C-7880. This paper reflects only the view of the authors and the funding bodies are not responsible for any use that may be made of the information it contains.


