This Greedy Piggy Went to the Ad Market: Stealing Users' (Input) Data using Mobile Sensors

Serafeim Moustakas

Thesis submitted in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes Campus, Heraklion, GR-70013,Greece

Thesis Advisors: Prof. *Evangelos Markatos*, Associate Prof. *Sotiris Ioannidis*

Heraklion, February 2021

This work was partially supported by **Institute of Computer Science, Foundation of Research and Technology Hellas**

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

This Greedy Piggy Went to the Ad Market:
Stealing Users' (Input) Data using Mobile Sensors

Thesis submitted by
Serafeim Moustakas
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author:

Serafeim Moustakas

Committee approvals:

EVANGELOS MARKATOS
Digitally signed by EVANGELOS MARKATOS
Date: 2021.03.23 21:24:31 +02'00'

Evangelos Markatos
Professor, Thesis Supervisor

SOTIRIOS IOANNIDIS
Digitally signed by SOTIRIOS IOANNIDIS
Date: 2021.03.29 12:57:29 +03'00'

Sotiris Ioannidis
Associate Professor, Thesis Co-Supervisor

Vassilis Christophides
Professor

Departmental approval:

Polyvios Pratikakis
Digitally signed by Polyvios Pratikakis
Date: 2021.04.15 10:55:06 +03'00'

Polyvios Pratikakis
Assistant Professor, Director of Graduate Studies

Heraklion, February 2021

Abstract

Mobile sensors in modern smartphones play a crucial role in the human-computer confluence by enhancing and transforming the user experience. However, misuse of mobile sensors combined with the absence of sufficient access control mechanisms introduce a plethora of privacy and security risks. As previously demonstrated, there is a wide range of sensor-based attacks using the rich data captured from mobile sensors and while previous attack paths depended on specific requirements such as malware or visiting a webpage; we found that an alternative and stealthier approach exists and affects *all* Android users without *any* requirements.

In this thesis we introduce a novel attack channel, that abuses the advertising ecosystem for delivering a variety of sophisticated and sneaky attacks using mobile sensors. The proposed threat-model does not depend on app permissions or user specific actions and affects all Android apps that contain in-app advertisements due to improper access control for sensor data in WebViews. We explain how motion sensor data can be used to infer user's sensitive touch input (pin, password, credit card info, etc.) in two distinct attacks scenarios, namely intra and inter-app data exfiltration. The former targets information obtained from the app that display the in-app ads, while the latter targets *every* other Android app installed on the device. Unfortunately, as in-app ads have the ability to "piggyback" on the permissions obtained for the app's core functionality they can also obtain information from other sensors such as the camera, the microphone and the GPS. To provide a comprehensive assessment of this emerging threat, we conduct a large-scale, end-to-end, dynamic analysis of in-app ads that access mobile sensors in applications found in Google Play. We find that in-app ads access and leak data obtained from motion sensors in the wild and emphasize the need for a strict access control policy that should be adopted and standardized to better protect users and the advertising ecosystem.

Thesis Supervisor: Prof. Evangelos P. Markatos
Thesis Co-Supervisor: Associate Prof. Sotiris Ioannidis

Περίληψη

Οι ενσωματωμένοι σένσορες στα κινητά τηλέφωνα έχουν ένα πολύ σημαντικό ρόλο στην συμπόρευση ανθρώπου-υπολογιστή, ενισχύοντας και αλλάζοντας ριζικά την εμπειρία του χρήστη. Ωστόσο, η κακή χρήση τους σε συνδυασμό με την απουσία επαρκών μηχανισμών ελέγχου πρόσβασης παρουσιάζουν μια πληθώρα κινδύνων προσωπικού απορρήτου και ασφάλειας. Όπως έχει ήδη αποδειχθεί, υπάρχει ένα ευρύ φάσμα από επιθέσεις σε κινητές συσκευές χρησιμοποιώντας τα "πλούσια" δεδομένα από τους ενσωματωμένους σένσορες. Ενώ σε προηγούμενες έρευνες αυτές οι επιθέσεις χρειάζονταν μια κακόβουλη εφαρμογή εγκατεστημένη στην συσκευή ή την επίσκεψη μιας κακόβουλης ιστοσελίδας, παρατηρήσαμε ότι υπάρχει μια διαφορετική, πιο αθόρυβη προσέγγιση που επηρεάζει όλους τους Android χρήστες χωρίς καμία προϋπόθεση.

Σε αυτή την εργασία παρουσιάζουμε ένα εναλλακτικό κανάλι επίθεσης, που χρησιμοποιεί το οικοσύστημα της διαφήμισης για να παραδώσει ένα μεγάλο πλήθος από διαφορετικές επιθέσεις χρησιμοποιώντας σένσορες σε κινητά τηλέφωνα. Το συγκεκριμένο μοντέλο απειλής δεν βασίζεται σε συγκεκριμένες "άδειες" εφαρμογών και επηρεάζει όλες τις Android εφαρμογές που δείχνουν διαφημίσεις λόγω του ακατάλληλου μηχανισμού πρόσβασης στους σένσορες. Αναλύουμε τον τρόπο με τον οποίο οι σένσορες κίνησης μπορούν να αποκαλύψουν προσωπικούς κωδικούς και πληροφορίες για την πιστωτική κάρτα σε δύο διαφορετικά σενάρια. Το πρώτο στοχεύει να αποσπάσει πληροφορίες από την εφαρμογή που προβάλει τις διαφημίσεις ενώ το δεύτερο στοχεύει όλες τις εφαρμογές που είναι εγκατεστημένες στο κινητό ενός χρήστη. Το συγκεκριμένο μοντέλο επίθεσης μεγαλώνει καθώς οι ενσωματωμένες διαφημίσεις στις εφαρμογές μπορούν να χρησιμοποιήσουν και άλλους σένσορες όπως την κάμερα, το μικρόφωνο και το GPS αν η εφαρμογή διαθέτει την κατάλληλη "άδεια". Αξιολογούμε την συγκεκριμένη απειλή, διεξάγοντας μια μεγάλη έρευνα στις διαφημίσεις που προβάλλονται από τις εφαρμογές του Google Play. Τα αποτελέσματά μας δείχνουν ότι οι διαφημίσεις παίρνουν πρόσβαση στους σένσορες κίνησης και διαρρέουν τα αντίστοιχα δεδομένα χωρίς καμία συναίνεση από τον χρήστη. Η έρευνά μας αποδεικνύει την επιτακτική ανάγκη ενός αυστηρού μηχανισμού ελέγχου πρόσβασης στους σένσορες των κινητών συσκευών για την προστασία των χρηστών και του οικοσυστήματος της διαφήμισης.

Επόπτης: Καθηγητής Ευάγγελος Π. Μαρκάτος
Συνεπιβλέπων: Αναπληρωτής Καθηγητής Σωτήρης Ιωαννίδης

Acknowledgments

Στους γονεις μου

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ubiquitous nature of mobile devices and the plethora of rich functionalities they offer has rendered them an integral part of our daily routines. The advent of smartphones has also transformed how users experience and interact with web services, and Android has become the most prevalent mobile operating system as it currently powers over 85% of devices worldwide [1]. Android's app ecosystem is dominated by free apps, and in-app advertisements have become the defacto source of revenue for app developers [2, 3]. Even major tech companies' ad revenue depends on mobile advertising, with Facebook earning 94% of its total ad revenue from mobile devices [4].

Recently, mobile motion sensors (e.g., accelerometer and gyroscope) have started playing an increasingly important role in the mobile advertising ecosystem, as motion-based ads allow for more interactivity and higher user engagement, leading to increased revenue [5]. Even though mobile sensors provide functional diversity that is reshaping how users interact with and consume ads, they also introduce a significant security and privacy threat. In more detail, a plethora of prior studies have demonstrated that data obtained from mobile sensors can be used for identifying and tracking users across the web [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23], inferring physical activities [12, 24, 13, 14, 25] and in more severe scenarios inferring users' touch screen input [25, 26, 27, 28, 29, 30]. Das et al., [31] also demonstrated that web scripts accessing mobile sensors allow for stateless tracking on the mobile web, while Marcantoni et al. [32] described how a plethora of mobile sensor-based attacks that previously required a malicious app to be installed can easily migrate to the mobile web using the HTML5 WebAPI.

However, as mobile users spend the majority of their browsing time within mobile apps [33], mobile ads will often reach their audience through in-app advertisements. These ads are shown inside the context of a mobile application and allow developers to release their apps for free while earning revenue from the embedded ads. Unfortunately, this symbiotic relationship combined with ads' ability to access mobile device sensors creates a novel and stealthy attack vector for delivering a variety of sensor-based attacks. Moreover, due to the nature of in-app ads and the complex processes (i.e., real-time bidding) that take place before reaching the user's device, further complicates the process of enforcing the security checks (e.g., WebView Cross-App Scripting) that the Android app store imposes

on apps before being published [34].

While prior work has proposed separating the privileges offered to applications and advertisements [35], Android has not adopted such an approach. To make matters worse, mobile motion sensors are *not* guarded by a specific permission and are freely accessible to in-app ads. Comparatively, the iOS operating system blocks in-app adds from accessing motion sensors or explicitly requests user approval when websites attempt to access them. To the best of our knowledge no prior study has explored the security risks posed by Android's access control and permission system policies that govern how in-app advertisements (and by extension other WebView objects) can use mobile sensors.

In this thesis we introduce a novel attack channel that abuses the ad ecosystem for delivering a variety of sophisticated and stealthy attacks. We present our threat model in which a malicious actor performs a "seemingly" legitimate mobile ad campaign, targeting genuine mobile apps downloaded from the official Play Store and by using the rich data returned from motion sensors is able to perform a plethora of sensor-based attacks including stealing login credentials and credit card information. We investigate how our threat model can be used to exfiltrate data in two distinct attack scenarios namely intra and inter-application data exfiltration. In the intra-application attack scenario, a motion based ad is able to capture credentials when ads are shown in Android `Views` that contain sensitive input information. Even though it is a good security practice to not show ads in sensitive `Views`, we found that for our threat model this specific dependency is not always required. Specifically, Google's interstitial ad placements can be easily misused for capturing sensitive input even if they are not displayed on top of sensitive `Views`, since JavaScript in interstitial ads is executed from the moment the advertisement is preloaded and up until the user clicks the corresponding application element. Furthermore, our attack vector increases exponentially in the inter-application attack scenario, since legitimate applications (e.g., Viber) that hold a specific Android permission for their core functionality permit ads to execute JavaScript in the background, therefore making *every* Android app installed on the device vulnerable to *all* sensor based side-channel attacks, including credential theft. Our experimental methodology proves our initial assumption that in-app advertisements not only have the potential to access mobile sensors but are also able to silently leak such values. Due to the severity of these attacks, we build a novel automated framework that analyzes in-app advertisements in the wild, provides an in-depth view of requests to access mobile sensors and distinguishes sensor access requested by in-app advertisements from those requested by the app's functionality.

Overall, this thesis makes the following contributions:

- We introduce a novel attack channel, that abuses the ad ecosystem, for delivering a variety of sophisticated and stealthy attacks using mobile sensors (e.g., Accelerometer, Gyroscope, etc.) inside in-app advertisements. We focus on inferring the user's touch input due to the severity of stealing sensitive data, but we note that *any* other sensor-based attack is also feasible. Additionally, we analyze the presented threat model in two different attack scenarios, namely intra and inter-application data exfiltration. Ads displayed in the former attack scenario target the application that displayed these ads, while ads displayed in the latter make vulnerable *every*

application installed on the device.

- We conduct a set of targeted and carefully designed experiments of in-app ads accessing mobile *motion* sensor data, that also involve experiments with an ad campaign. Our results highlight the severity of accessing motion sensors inside in-app advertisements and motivate us to conduct a large-scale, end-to-end, automated study of in-app ads that access *all* mobile sensors across 4K of the most popular apps from the official Google Play. We analyzed our dataset multiple times between 09/01/2020 and 02/28/2021 using a realistic dynamic framework consisting of smartphone devices that provides an in-depth view of requests to access mobile sensors. Our framework uses several VPN sessions and our study is not bound by the ads displayed in a specific country.

- Findings reveal an emerging threat, since sensor information is obtained and leaked from in-app ads in the wild. Due to the severity of the attacks, we discuss and propose a set of guidelines for access control policies that should be adopted and standardized to better protect users. Information and vulnerabilities indicated in this thesis, will be responsibly disclosed to Google and to the developers of the affected apps.

Chapter 2

Background Info & Motivation

This section provides background information about the ad ecosystem, discusses previous work about mobile sensor-based attacks, provides technical information for displaying in-app ads and describes possible pitfalls that can affect the user's privacy.

Ad Ecosystem. The advertising ecosystem is a massively complex and hard to map area, consisting of different entities. The major key players of the ecosystem are the Demand side and the Supply side platform. The Demand Side Platform (DSP) is utilized by brands, media agencies and trading desks and allows them to message their consumers across various channels. On the contrary, the Supply Side Platform (SSP) consists of apps and publishers on the web and coordinates and manages the supply and distribution of ad inventories. In general, SSPs allows sites and apps to make their ad placements accessible from the ad ecosystem, while DSPs give the opportunity to brands, agencies and tradedesks to run and target their advertising campaigns. The collaboration between DSPs and SSPs happens with Ad Exchangers and Ad Networks. Ad Exchangers provide the real-time bidding functionality of the ad ecosystem where the higher bidder from the DSP side buys an ad placement of the SSP for a specific site. Ad Networks offer similar functionality but with a more direct communication between publishers and advertisers. Other players in the ad ecosystem are Ad Servers and Data Management Platforms. Ad Servers are responsible for converting a simple image into an interactive HTML5 online ad, that can retrieve metrics for the advertiser such as clicks, viewability and engagement level. Finally, Data Management Platforms are necessary for targeting ads to a relative audience and identifying new customers based on their interests and demographic characteristics.

Mobile Sensors. A plethora of research papers (e.g., [14, 36, 37, 38, 39, 40, 41, 7, 18, 42]) have demonstrated that the rich data acquired from sensors such as the Accelerometer, the Gyroscope and the Light sensor can be used for a plethora of sophisticated attacks, with accuracy reaching up to 94% in certain attack scenarios [25], and without requiring any permission from the operating system or the user. Researchers have previously presented a taxonomy of sensor-based attacks [43, 32], where attacks are classified in four major categories; Physical Activity Inference, Acoustic Attacks, Digital Activity Inference and User Tracking. A notable example that reveals the severity of such attacks is the Touchscreen Input Attack of the Digital Activity Inference category that shows how sensor

information (including the Accelerometer and Gyroscope) can be used to infer what the user is typing on the smartphone's touchscreen (e.g., [25, 26, 28, 29, 44, 45, 27, 46, 47]). This attack is made possible by the changes in the screen's position and orientation, and the motion that occur while the user types.

Ads, WebViews & Sensors. Advertisements are usually written in JavaScript which enables the use of a plethora of powerful API calls. Amongst these API calls are the HTML5 functions responsible for accessing mobile motion sensors. Specifically the accelerometer sensor is accessed using the `DeviceMotionEvent.acceleration` [48] and the `DeviceOrientationEvent` [49] APIs, while the `DeviceMotionEvent.rotationRate` [50] API gives access to the gyroscope sensor. Moreover, the Generic Sensor API [51] bridges the gap between native and web applications and can be easily extended with new sensor classes with very few modifications. As mentioned in [52], since this API is not bound to the DOM (nor the Navigator or Window objects) it opens up many opportunities including its use within service workers. Even though the features of this API have not yet been finalized, we tested the Generic Sensor API and we found (at the time of writing) that web and service workers do not have access to motion sensors. Nonetheless we argue that if this feature is considered to be implemented in the future, it should be done carefully and coupled with the permissions API [53].

Previous work [31, 43] found that many websites and third-party scripts access the information provided by these sensors when accessed through a mobile browser. In practice, the mobile advertising ecosystem has two different paths for displaying advertisements to users, either through the advertisements that are embedded in a website that is accessed using a mobile browser or through in-app advertisements. The latter are displayed inside the context of a mobile application with the use of an Android WebView [54]. WebView is based on the Chromium project and WebView objects are able to display web content as part of an activity layout. Even though WebView may lack some of the features of full-fledged browsers, it can evaluate JavaScript (e.g., `evaluateJavascript()`), interact with cookies (e.g., `setCookie()`/`getCookie()`) and can access a plethora of mobile HTML5 APIs. Additionally, since WebView exists in the same address space as the actual application's process, it also shares all of the application's privileges (includes normal and dangerous permissions). To verify that this occurs in practice, we conducted an exploratory experiment with an Android WebView accessing mobile HTML5 APIs. We created a mock app and separately executed all HTML5 APIs that access mobiles sensors. We found that WebViews are able to call a plethora of mobile sensors. Moreover, we found that all mobile sensors (except GPS and Camera) do not require the host application to hold an Android permission. Furthermore, if the application holds the appropriate permissions then WebView automatically and without any interaction gains access to these resources. For example, if the app has the `ACCESS_FINE_LOCATION` permission, then the WebView can capture the GPS values. This simple experiment highlights how Android offers functionality that can be misused by in-app advertisements and can affect user's privacy. To make matters worse, the ability that ads have to "piggyback" on the permissions obtained for the app's core functionality, enables an attack vector that can be used to deliver *any* sensor-based attack and rendering *every* app installed on the device vulnerable. This occurs due to the `SYSTEM_ALERT_WINDOW` permission which we describe in Section 3.2.

Chapter 3

Threat Model & Attack Scenarios

This section introduces our threat model, describes preliminary experiments involving advertising campaigns and analyzes how these experiments prove our initial hypothesis. We provide technical details on how to exfiltrate data in two distinct attack scenarios namely the *intra-app* and *inter-app* data exfiltration.

3.1 Threat Model

In this thesis we propose a new channel that abuses the mobile advertising ecosystem for delivering a mobile sensor-based attack which affects every Android device (71.93% of mobile users worldwide [55]). Contrary to other attacks, the proposed threat model does not require any malicious application installed or the device since every application that is showing advertisements is susceptible, while users are completely unaware. The attack uses in-app advertisements that access the motion sensors in order to silently steal any information that is typed on the screen, including login and passwords credentials as well as credit card information and pin numbers. Even though we selected inferring the user's input as our proof of concept attack, which is also the most frequently feasible attack as described in [43], our attack scenario can be tailored for any sensor-based attack. This is possible due to the lack of any restriction in accessing the device's sensors (except Camera and Mic) either through an Android permission or a user prompt. As can be seen in Figure 3.1, a malicious actor creates a legitimate advertisement that intentionally accesses the device's motion sensors and pays for a mobile ad campaign. Since ad campaigns can be tailored for specific needs, the malicious actor informs the Ad Server or the DSP, that his advertisement should only be displayed in mobile devices, specifically as an in-app advertisement (and/or if needed in specific apps), improving significantly his attack vector. The actual context of the advertisement does not really matter but can be tailored for more impressions based on recent trends (e.g., Covid vaccine, elections, etc.) The advertisement follows the procedure of publishing a mobile advertisement and eventually is displayed as an in-app advertisement in different applications. When the advertisement reaches the user's device, the JavaScript code performs the appropriate HTML5 API calls for accessing the motion sensors and then leaks this data to a server controlled by the malicious actor. We

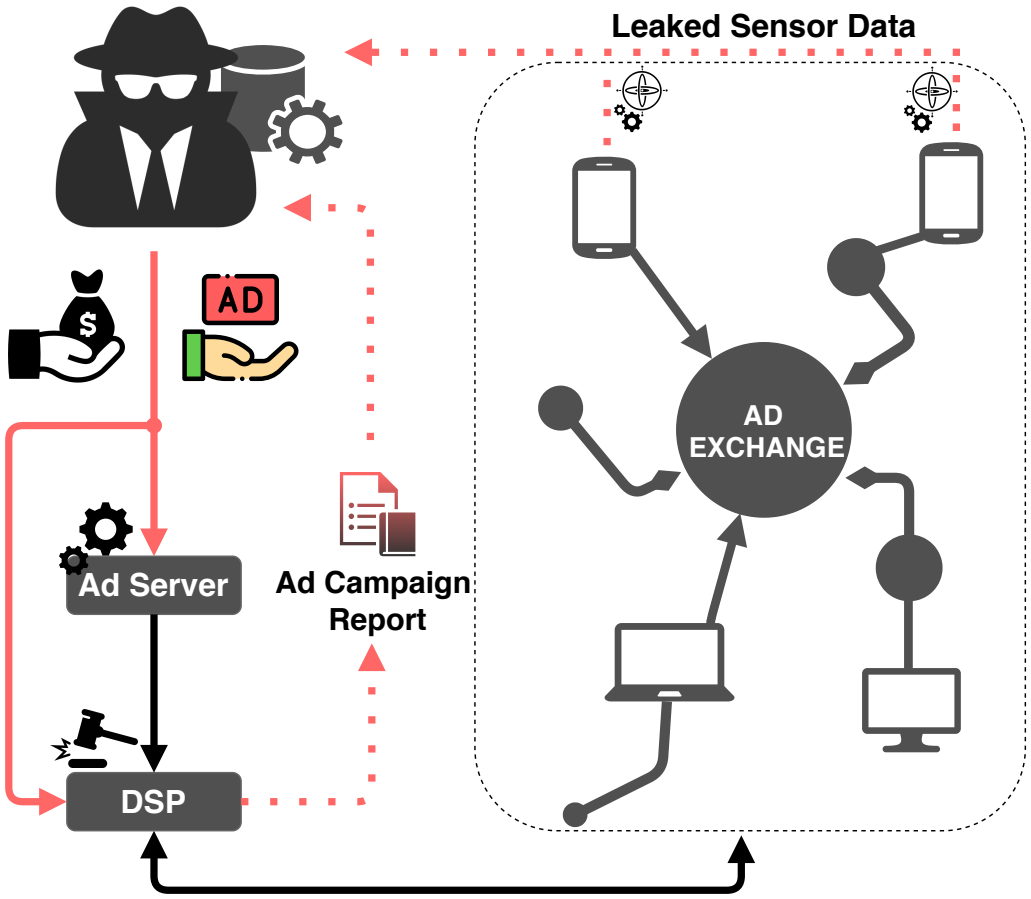

Figure 3.1: Threat model. A malicious actor abuses the advertising channel, by paying for an advertising campaign in order to publish a "seemingly" legitimate mobile in-app ad that silently captures and leaks sensor information.

note that access to motion sensors by in-app ads does *not* require any interaction from the user (i.e., clicks). Since the malicious actor can access real users' motion data using in-app ad campaigns, we briefly discuss how such data can be used to reconstruct the user's input in order to steal sensitive information such as login credentials and credit card information.

Previous work (e.g., [25, 26, 28, 29, 44, 45, 27, 46, 47]) in this field proposed techniques for recreating the user's touch screen input from motion sensors with results achieving up to 94% in accuracy [25]. The attacker needs to implement this tool once in order to reconstruct the touch input from motion sensors; and preprocess the data to include only relevant information, such as emails, passwords and credit cards. Re-implementing one of the many attacks that have been demonstrated using sensor data is out of the scope of our work and apart from the details included in those studies, even tutorials and tools are publicly available [56, 57]. Our focus is on revealing a novel attack vector that magnifies the impact and scale of such attacks, and allows attackers to stealthily reach millions of

Table 3.1: Feasible intra and inter-app data exfiltration scenarios of in-app ads that access sensors. CAM, MIC and GPS requires that the app holds the appropriate permissions. For devices running API > 28, apps also require `android.permission.ACCESS_BACKGROUND_LOCATION` for accessing GPS in the background. API 30 allows different options for dangerous permissions. We tested the permission option "Allowed only while in use" (P.O.1) for CAM and MIC. For GPS we tested "Allowed only while in use" (P.O.1) and "Allowed all the time" (P.O.2).

	Motion	**CAM** P.O.1	**MIC** P.O.1	**GPS** P.O.1 | P.O.2
without `SYSTEM_ALERT_WINDOW`	Intra	Intra	Intra	Intra
with `SYSTEM_ALERT_WINDOW`	Inter	Inter	Inter	Inter

devices without the need for a malicious app to be downloaded or users to be tricked into visiting a malicious page.

3.2 Intra & Inter-Application Attacks

Here we provide technical details about two distinct attack scenarios that can be used to exfiltrate sensitive data from an Android device, namely intra and inter-application data exfiltration. We also briefly discuss other feasible sensor-based attacks, besides touch input inference that fall in the same scope. Table 3.1 summarizes app permission requirements (if any) and sensor accesses in each scenario. We also include after which user actions, sensor access is still granted for the inter-application data exfiltration scenario.

Intra-Application Data Exfiltration. In this scenario we can capture the input data of the Android app that is displaying the advertisement that accesses the motion sensors. This can be done with two ways which we describe below or using a combination of both. Advertisements are displayed inside an Android `WebView` which is responsible for loading all the elements of the advertisement from the web. Each WebView is attached to an Android `View`, which is responsible for displaying the application's content on the screen. At the time the WebView has finished loading the content of the advertisement, the appropriate HTML5 APIs are executed and the advertisement can capture touch input from the `View` that is attached to it. This is extremely important since many `Views` in Android apps contain sensitive input. We note that besides the WebView responsible for displaying the ad, other WebViews may exist in the same `View` for different functionality such as login or payments. Therefore, any part of the application that is attached to the `View` that contains the advertisement is vulnerable for input hijacking. Even though it is a good practice to not show ads in `Views` with sensitive input, in our analysis we found certain cases of such bad practice. Interestingly, the attack space increases if the application is using Google's interstitial ad placements. Interstitial ads are able to preload the content of the advertisement before it is displayed on the screen and we found that

Google's library for interstitial ad placements, allow the interstitial ad to execute code from the time they are preloaded and until the user has closed the advertisement. Since an interstitial ad will be displayed only when a specific element of the app is pressed (and they can be placed on any element) the code of the advertisement will be executed until this specific element is pressed. As such, the user may explore other parts of the app, including `Views` with sensitive content, while the interstitial ad is capturing values from motion sensors. It is worth noting that loading the interstitial ad (i.e., `loadAd()`) as early as possible to ensure it is available during the `show()` function, is also encouraged by the developer documentation [58]. Furthermore, our experiments with Google's library for interstitial ad placements reveal that these ads continue to execute code not only in different `Views` but also in different `Activities` within the same app and even if the application `Activity` that initiated the preloading mechanism has been destroyed (e.g., `activity.finish()`). As such, interstitial ads not only increase the robustness of intra-application data exfiltration, but also deceive users since they feel safer when a sensitive `View` does not display ads. As we discuss in our analysis we found a great number of apps using interstitial ads.

Inter-Application Data Exfiltration. Android apps are executed in a sandbox environment and in different processes to prevent unintended data leakage from one app to another. WebViews by default are attached to the app's UI thread and can not execute code in the background if the user switches applications. Surprisingly, Android offers mechanisms to execute code in the background, by attaching a `View` in the `WindowManager`, and the same applies for WebViews. We found that if the host application holds the `SYSTEM_ALERT_WINDOW` permission for it's core functionality, then any ads displayed in this app can use the permission to place the WebView in the background and continue accessing motion sensors even if the user switched applications. The `SYSTEM_ALERT_WINDOW` permission, as mentioned by the official Android SDK [59], falls into a special category of permissions that require the user to explicitly grant it when requested (the app opens the Android Settings for this specific app and informs the user of the permission's abilities). Interestingly, if applications are downloaded directly from the official Google Play, then this permission is granted automatically and without any user interaction [60, 61]. We argue that such cases of relaxed policies, not only confuse users and developers alike but can be misused for devastating attacks. Furthermore, even experienced users that can identify suspicious apps that were automatically granted the permission can be mislead. This is especially true for popular apps that need this permission for showing pop up messages and providing additional functionality on top of other apps. Applications requesting this permission include but not limited to Skype, Facebook Messenger and Viber. We note that Viber a very popular messaging app, also used by banks for two-factor authentication messages, contains ads and is susceptible for the inter-application data exfiltration. In order to understand the magnitude of this attack scenario, we note that if one application holds this specific permission and is displaying ads, *all* apps installed on the device can be compromised and are vulnerable to input hijacking. Even banking apps that use the `WindowManager.LayoutParams.FLAG_SECURE` option, a security feature to treat the contents of the window as "secure" [62] are vulnerable from sensor-based inter-app side channel attacks. As we describe later on, we found 410 apps in our dataset that hold this

permission, with 291 of them also displaying ads. We tested this functionality on two Pixel 4 devices running (AOSP) API 29 and API 30 respectively. The device running Android Version 11 (API 30), at the time of writing had the latest security updates (February 5, 2021).

Feasible Sensor-based Attacks. There are many other sensor-based attacks that can be exploited using the proposed threat model besides touch input inference. Digital fingerprinting, physical trait or demographic inference, speech recognition, inaudible communication are just a few examples and the possibilities for such attacks when deployed using ads are endless. A comprehensive list with sensor-based attacks can be found in [43].

3.3 Hypothesis Testing

In order to identify whether in-app advertisements delivered through the ad ecosystem can access and leak data from motion sensors, we conducted a set of preliminary experiments. Due to the severity of the attacks we present, and the ethics involved with our work, we design these experiments carefully. Our goal is to prove our initial hypothesis without putting any subjects at risk.

Experiment #1. In this experiment we verified that in-app advertisements are able to access motion sensors and leak these values using common network techniques. We set a Raspberry Pi posing as an Ad Server and deployed our advertisement to a mock-up application in our own device. We performed our experiment using a Nexus 5x device and verified that the ad was successfully displayed, accessed the motion sensors and using either an `XMLHttpRequest` or a `GET/POST` method, leaked the sensor values to the Raspberry server. We performed this experiment twice; once for sending an one-time value and once for sending continuous values since these sensors use an event listener to continuously fire events.

Experiment #2. In this experiment we verified that an in-app advertisement accessing motion sensors can be delivered to users through the complete process of publishing an advertisement, including any real-time bidding process. As such we contacted a Demand-Side Platform (DSP) and informed them that we would like to buy a mobile in-app ad campaign to display an advertisement that also makes use of the motion sensors. In this experiment we did not gather any information (including sensor values) from users. We signed a paid contract with the DSP and during the ad campaign, we searched for our advertisement by manually exercising different apps. Surprisingly, we found that the in-app motion-based ad campaign performed better than a regular ad, since the performance on the engagement level was 3-4 times higher than the general benchmark of simple banner campaigns. The report returned from the DSP contained aggregated results of the ad campaign (e.g., apps displayed, impressions, clicks, etc.) which can not be used for any sensor-based attack including device or user fingerprinting.

Lessons Learned. Our experiments verify that the mobile ad ecosystem publishes in-app ads that access motion sensors. On the contrary, we can not verify that such values can also be leaked. Any experiment that collects sensitive information from actual users involves many moral and ethical issues. Interestingly, during our preliminary analysis we

identified a Vodafone ad that accessed motion sensors without any interaction from the user and leaked these values, with a `GET` request, to a DoubleVerify domain. Since DoubleVerify provides online media verification and campaign effectiveness solutions, we believe that such practices are used for bot detection and ad fraud prevention. Even though we can not confirm nor deny any malicious intentions behind this case, we believe that users should be given the option to allow or deny access. Motivated by our findings we believe that it is possible for anyone to abuse the mobile ad ecosystem for exfiltrating data by delivering a sneaky advertisement that captures the rich information provided by these data. It is important to note that different Ad Networks and DSPs may have different policies about the JavaScript code of an advertisement or they may not care at all. Additionally, Ad Networks and DSP's may dynamically analyze in a sandboxed environment the functionality of an ad before publishing, as to eliminate cases of malvertising. Understanding how relaxed or not is the mobile advertising ecosystem about the ads it serves or finding techniques to bypass any mechanisms employed for malvertising, falls outside the scope of this thesis. Our goal is to investigate whether the mobile ad ecosystem is being abused to deliver sophisticated and stealthy attacks that amongst others can steal sensitive input information such as login credentials and credit card information.

3.4 Ethical Considerations

In the experiment with the ad campaign we did not gather any information that can be used to identify/deanonymize or harm users in any way and the only information available to the authors was the report returned by the DSP, containing aggregated results about the performance of the advertisement. Moreover, the framework for analyzing apps does not click on advertisements.

Chapter 4

Analyzing in-app ads in the Wild

Motivated by the severity of mobile motion sensor-based attacks, the absence of *any* access control mechanism (neither through an Android permission or a user prompt) for access to these sensors and the results of our preliminary experiments, we conducted a large-scale, end-to-end automated study of in-app advertisements accessing mobile sensors. We dynamically analyzed applications with in-app advertisements and monitor access to *all* available mobile sensors and record any potential leakage of this type of data. It is important to note that the Accelerometer, Gyroscope, Light, Magnetometer and Proximity sensor, do not require any kind of permission (as is the case with Camera and Mic). In Section 4.1, we present our framework and methodology for analyzing in-app ads inside Android applications. We give an overview of our architecture and provide implementation details about the in-line hooking methods for intercepting both JavaScript and Android system call functions. Section 4.2 presents our application dataset and Section 4.3 presents the results of our in-app advertisement study.

4.1 Framework Details

We obtain an in-depth view of sensor data access and distinguish sensor access requested by the in-app advertisements from those requested by the app's functionality, by monitoring *all* sensors' access at two different layers using multiple components. As can be seen in Figure 4.1, for each of these layers (Android and Network) we monitor different API calls using modules of the Xposed framework [63] and injected JavaScripts respectively. Our testbed consists of three Nexus 5x devices, running Android 7.1.1 that we configured with `mitm`'s certificate in order to intercept HTTP/S traffic.

Android Layer. The Android layer (Figure 4.1 - left) monitors applications' access to sensors by intercepting Android system calls using a custom Xposed module that detects and hooks requests to sensor-specific Android API calls. Since values returned by these sensors can be used by apps to evade analysis or hide suspicious activity [64], we made our infrastructure more realistic by also intercepting the values returned by certain sensors and slightly modify them within the appropriate and legitimate bounds. Advertisement hyperlinks inside WebViews, are identified by hooking the appropriate WebView and

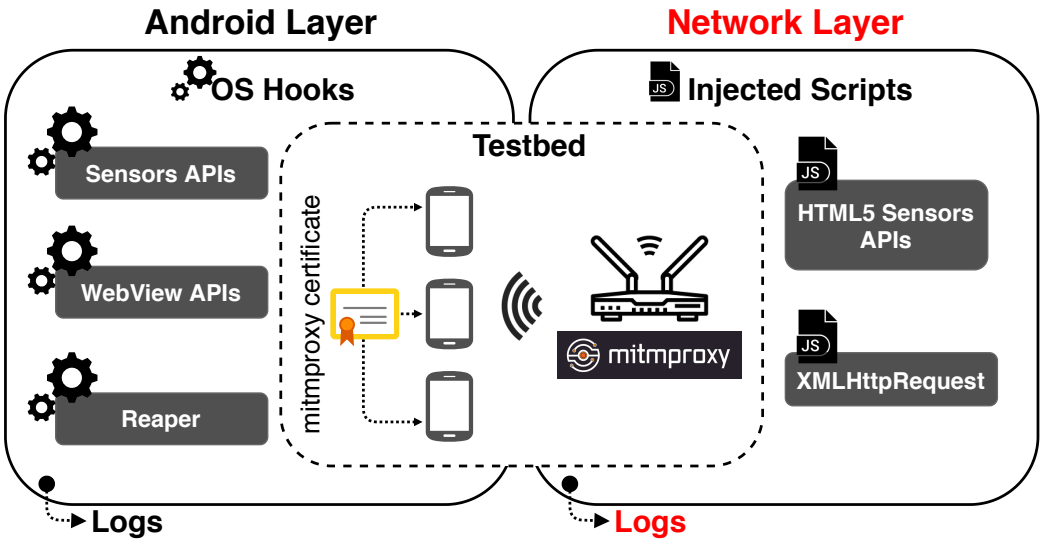

Figure 4.1: Overview of our framework's infrastructure. The combined components of both layers provide an in-depth view of requests to access mobile sensors and distinguish sensor access requested by the in-app advertisements from those requested by the app's functionality. Components in the Android layer (left) are responsible for monitoring system API calls, while components in the Network layer (right) monitor JavaScript calls and network traffic.

Chromium APIs. Additionally, in this layer we use the functionality offered by the Reaper [65] for (i) verifying which of the sensor-specific Android API calls are permission-protected and (ii) traversing the application's graph using a breadth-first traversal for achieving high traversing coverage.

Network Layer. At the network layer (Figure 4.1 - right) our framework employs a transparent proxy server that intercepts all network traffic by using `mitmproxy` [66] and injects the appropriate JavaScript components for intercepting JavaScript calls. We used the javascript-hooker Node.js module [67] that allows us to hook any JavaScript function called inside a WebView and intercepts the method to be called and its arguments. Using this approach (i) we hooked all the functions that access and retrieve mobile-specific sensor data through the official mobile HTML5 WebAPI [68] and (ii) we monitor JavaScript calls of the `XMLHttpRequest` function, since in-app advertisements can also leak data using this approach. Table 1 in Appendix 7, lists all monitored HTML5 WebAPIs.

By combining hooks both from low-level sensor function calls of the operating system and JavaScript calls from the network, we can successfully distinguish sensor access requested by the in-app advertisements from those requested by the app's functionality. Specifically, if we identify a sensor system call from the OS without the same sensor API call from the network, then the functionality of the application requested access to this sensor. On the contrary if we identify both a sensor call (e.g., for the Accelerometer) at the network layer and at the Android layer then we can successfully deduce that the in-app

advertisement accessed the mobile sensor. It is worth noting that in cases where both the application and the in-app advertisement performs the same sensor call it does not affect the results of our analysis. Our goal is to find ads that access mobile sensors and we flag the advertisements that access mobile sensors only if we find the appropriate function calls in both the Android and Network layer. Finally in order to avoid conflicts with other apps accessing sensors, we analyzed each app individually and limit any other background app activities using the adb toolkit. We verified that our framework behaves as expected by performing again the Experiment #1 (see Section 3.3) for all mobile sensors.

4.2 Dataset & Experimental Setup

Our application dataset consists of 3948 applications in total. We downloaded free apps from Google Play using the Raccoon [69] framework. Overall, we selected the top 100 apps (or as many as were available) from each category, and downloaded a total of 3946 from 61 categories. Since previous studies [31, 43] provided a list of websites that access mobile sensors, for each one of them, we tried to find its corresponding mobile app (if it exists in the official store) and included it in our dataset. Additionally, adult websites as mentioned in [43] also access a plethora of mobile sensors and such data can be used to track users across and outside the adult web [70]. Therefore, we also included such applications in our dataset. As applications with pornographic content can not be found in the official store, we manually download them from their corresponding website.

Since we can not know when a specific ad campaign that accesses motion sensors is running, nor can we know which app will trigger this specific advertisement, we opted for a large number of unique apps from different categories that were analyzed multiple times over the course of six months. Furthermore, our infrastructure uses VPN to fetch ads from different countries and our study is not bound by the ads displayed in a specific country. For the experiments performed over VPN we selected a subset of 200 apps and analyzed them in several countries. Even though there are techniques to identify whether an app is hiding behind a VPN (e.g., GPS coordinates, nearby WIFI access points, etc.), we found that this simple approach is enough to deliver foreign ads. In general, we analyzed 3948 apps locally and 200 apps for each VPN session in other countries. In general our study includes advertisements from USA, Russia, India, United Kingdom, Germany and Greece. All applications are given at installation time all the permissions that they may request (including run-time permissions) using the "`adb install -g`" option.

4.3 Large Scale In-app Ad Measurement Study

Here we present our findings from our large-scale study on the use of HTML5 WebAPI calls by in-app advertisements in the wild.

WebAPI Accesses. As can be seen in Table 4.1, in-app advertisements access a plethora of HTML5 WebAPIs, mobile specific or not, across all countries. We found several instances of in-app advertisements accessing motion sensors using the `addEventListener(devicemotion)` or `addEventListener(deviceorientation)` We-

Table 4.1: Number of apps containing in-app ads accessing WebAPIs, analyzed over different countries.

WebAPI	#Apps per country					
	US	RU	IN	UK	DE	GR
Mobile						
window - devicemotion	3	2	2	1	2	9
window - deviceorientation	0	1	0	1	0	1
window - orientationchange	2	1	1	3	1	17
screen - change	0	0	0	1	0	1
getBattery	0	0	1	2	2	10
non-Mobile						
XMLHttpRequest.send	2	3	1	1	80	913
getTimezoneOffset	4	2	1	2	81	954
toDataURL	0	0	0	0	0	7
getContext	3	3	0	1	7	44
WebGLRenderingContext	0	0	0	1	1	6
setItem	1	1	0	0	92	1,167
getItem	0	0	1	0	81	1,016
removeItem	1	1	0	0	92	1,145
key	0	0	0	0	0	14
createElement(canvas)	4	5	1	3	12	56

Due to space constraints the complete HTML5 API names can be found in Appendix.

bAPIs. Compared to `addEventListener(orientationchange)` that detects only the orientation mode of the device (i.e., portrait or landscape), they return continuous values from the Accelerometer and the Gyroscope. We did not find any in-app advertisements accessing the camera, the microphone or the GPS of the device, even though many of the tested apps had these permissions in their Manifest file and were allowed to use them during run-time. Concerning the GPS sensor, we believe that in-app ads use another non-intrusive way for getting an estimate location of the device by utilizing the `Date.prototype.getTimezoneOffset` function, since it returns the time zone difference in minutes from the current locale. We also observed several ads using the `navigator.getBattery` API. As described in [71] the battery status reveals information about the maximum capacity of the battery and can be used to effectively track users across the web. Moreover, we observe that in-app ads access functions that are known to be used for canvas fingerprinting, such as `HTMLCanvasElement.toDataURL`, `HTMLCanvasElement.getContext`, `document.createElement(canvas)` and the `WebGLRenderingContext`. Finally we observe that in-app ads read, write and delete data from the local storage using `getItem`, `setItem` and `removeItem` respectively. Even though we did not check for any suspicious behavior for in-app ads accessing local storage, as it falls outside the scope of our work, we believe that such functions should be used carefully since local storage is vulnerable to XSS attacks.

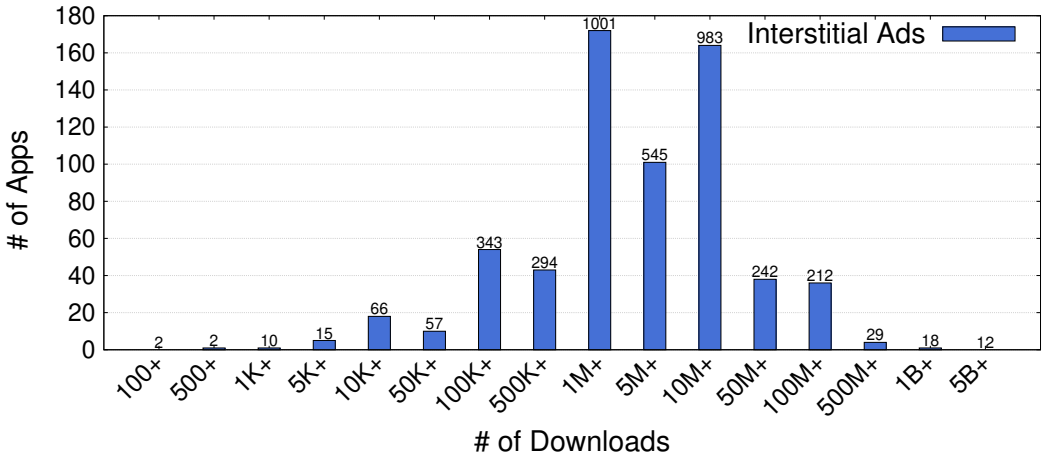

Figure 4.2: Number of apps with Google's interstitial ad placements. The number on the bars shows the total number of apps in each bucket. On average 17.05% of apps contain Google's interstitial ad placements.

Google's Interstitial Ad Placements. Interstitial ads are interactive advertisements that cover the interface of their host app. These ads appear between content or activities and allow for a more natural transition. As mentioned earlier, we found that Google's library for interstitial ad placements executes JavaScript from the time they are preloaded and until they are displayed and closed by the user. As such they increase the ability of ads to capture sensor data not only from the `View` that displays them but from others as well, increasing this way the attack vector for the intra-app data exfiltration attack. Our analysis shows that Google's interstitial ad placements exists on average on 17.05% of apps, independent of the number of downloads. Figure 4.2 shows the number of apps that contain Google's interstitial ad placements based on their numbers of downloads (i.e., popularity). The number on the bars shows the total number of apps analyzed in each popularity bucket. We observe that interstitial ad placements are more prevalent across apps that have between 100K+ and 100M+ downloads. Apps with 5B+ downloads are rare and most of them either do not contain ads (e.g., WhatsApp Messenger) or may use their own tools for interstitial ad placements (e.g., Facebook). We argue that no matter how popular an app may be, Google's interstitial ad library allows JavaScript to run before the ad is displayed on the screen and even if an app is securely configured, the embedded interstitial ads (at their current state) are a liability that can lead to privacy leakage.

Table 4.2: Top 30 most popular apps with the `SYSTEM_ALERT_WINDOW` permission marked by Google Play as "Contains Ads" (i.e., in-app ads), sorted in descending order based on the number of downloads. If one of these apps is installed on the device and a WebView is configured to run in the background, *all* of the user's apps are vulnerable to the touch input inference attack. Additional app permissions (CAM, MIC and GPS) allow in-app ads to silently capture photos, listen to conversations and retrieve the device's position even if the app is in the background.

⇓ DLs	Package Name	CAM	MIC	GPS
5B+	com.google.android.music	✗	✗	✗
5B+	com.facebook.katana	✓	✓	✓
1B+	com.lenovo.anyshare.gps	✓	✓	✓
1B+	com.twitter.android	✓	✓	✓
1B+	com.facebook.lite	✓	✓	✓
1B+	com.skype.raider	✓	✓	✓
500M+	com.imo.android.imoim	✓	✓	✓
500M+	jp.naver.line.android	✓	✓	✓
500M+	com.viber.voip	✓	✓	✓
500M+	com.mxtech.videoplayer.ad	✓	✗	✓
500M+	com.UCMobile.intl	✓	✗	✓
500M+	com.google.android.apps.youtube.music	✗	✗	✗
100M+	com.gau.go.launcherex	✓	✓	✗
500M+	com.opera.mini.native	✓	✗	✓
100M+	com.lbe.parallel.intl	✓	✓	✓*
100M+	com.playit.videoplayer	✗	✓	✗
100M+	com.digidust.elokence.akinator.freemium	✗	✗	✓*
100M+	com.apusapps.launcher	✓	✗	✓
100M+	com.jb.emoji.gokeyboard	✓	✓	✓
100M+	com.jb.gokeyboard	✓	✓	✗
100M+	com.ludashi.dualspace	✓	✓	✓
100M+	com.opera.app.news	✓	✗	✓
100M+	com.opera.browser	✓	✓	✓
100M+	com.rsupport.mvagent	✓	✓	✗
100M+	com.starmakerinteractive.starmaker	✓	✓	✓
100M+	in.mohalla.sharechat	✓	✓	✓
100M+	com.waze	✓	✓	✓*
100M+	ru.ok.android	✓	✓	✓
100M+	com.accuweather.android	✗	✗	✓
100M+	com.imo.android.imoimbeta	✓	✓	✓

*App also holds `android.permission.ACCESS_BACKGROUND_LOCATION`

`SYSTEM_ALERT_WINDOW` permission. As mentioned earlier, every app that requests this dangerous permission and is downloaded from Google Play automatically enjoys the permission's privileges without any user interaction or consent. This permission allows WebViews to be attached to the `WindowManager` and execute code that can access sensors in the background. We found that in our dataset 410 apps hold this permission and

291 out of them are marked by Google Play as "Contains Ads" (i.e., in-app ads). Table 4.2 shows the 30 most popular apps, sorted based on their number of downloads, that contain ads and hold this permission. Besides motion sensors that do not require a permission, for each app we included other dangerous permissions they hold and provide access to additional sensors (e.g., permissions for CAM, MIC and GPS) and can be abused by in-app ads. We note that if one of these apps is installed on the device and a WebView displaying advertisements is configured to run in the background (misconfigured intentionally or not, by the developer of the app or from an integrated third-party ad library), all of the user's apps are vulnerable to the touch input inference attack. Based on our findings we argue that these apps should carefully revise the security implications of this dangerous permission and whether it is really needed for their functionality. If the functionality provided by this permission is necessary, apps *should* inform users and ask for their consent.

Motion Sensor Leaks. During our experiments with in-app advertisements, we found several cases were motion sensors are accessed and the returned values are leaked to third-party domains. Table 4.3 presents these results with applications tested multiple times over several months. Each application that we list may have displayed more than one in-app advertisement accessing motion sensors during one execution (e.g., Vodafone ad). For each in-app ad that listens to `devicemotion` and `deviceorientation` events, since these APIs return continuous data, we have also marked whether the corresponding application is vulnerable to the intra or the inter-app data exfiltration attack. In the former attack, an app may be marked with a (◖) if it displays ads in sensitive `Views` (e.g., login), or with a (◗) if it uses Google's interstitial ad placements. If both cases are found they are marked with (●). In the inter-app data exfiltration attack, we mark all these applications that hold the `SYSTEM_ALERT_WINDOW` permission and give the ability to in-app ads to run in the background, making any other application installed on the device vulnerable to the touch screen input inference. For each entry we list the domain name of the ad placement. The last column denotes whether we identified any motion data leakage by manually analyzing the network traffic and the corresponding JavaScripts. We found that in most cases motion sensor values are leaked to DoubleVerify's domains. Entries that are not marked with sensor data leakage, means that we could not identify such values in the network, since most of the analyzed JavaScripts were heavily obfuscated, performed some form of transformation in the data and also used additional libraries downloaded from the network. We observe that in-app ads that access motion sensors are not bound by a specific country since in all of our VPN sessions we identified such cases. Moreover, in certain cases (e.g., com.genius.android) we found that apps display in-app ads with access to motion sensors independent of the origin country. The actual content of the in-app ads we analyzed varies and we found that the ads displayed during sensor accesses included, amongst others, Vodafone products, Disney+ promotions and casino online services. We observe that in many cases, apps displaying advertisements with access to motion sensors are vulnerable to one of the two presented attack scenarios, or in certain cases they are vulnerable to both.

Table 4.3: Non-browser apps with in-app ads that listen to `devicemotion` and `deviceorientation` events. We exclude in-app ads listening to `orientationchange` events as they only provide information about landscape or portrait mode. Intra-app vulnerability denotes that the app either displays ads in sensitive `Views` (◖) or uses Google's interstitial ad placements (◗). If both cases are found they are marked with (●). Inter-app vulnerability denotes that ads have the ability to run in the background using the app's `SYSTEM_ALERT_WINDOW` permission. Apps are sorted in descending order based on the number of downloads. Entries below rows with country names indicate findings from apps analyzed in these countries.

⇓ DLs	Package Name	Motion Events	Ori/tion Events	Intra Vuln.	Inter Vuln.	Ad Placement	Sensor Leaks
USA							
10M+	com.resultadosfutbol.mobile	✓	✗	◗	✗	pubads.g.doubleclick.net	-
5M+	com.genius.android	✓	✗	◖	✓	pubads.g.doubleclick.net	-
10K+	com.kdrapps.paokfcnet	✓	✗	◗	✗	pubads.g.doubleclick.net	✓
Russia							
5M+	com.genius.android	✓	✗	◖	✓	pubads.g.doubleclick.net	✓
1M+	com.studioeleven.windfinder	✓	✓	✗	✗	pubads.g.doubleclick.net	-
India							
5M+	com.bingoringtones.birds	✓	✗	◗	✗	pubads.g.doubleclick.net	✓
500K+	com.appscores.football	✓	✗	◗	✗	pubads.g.doubleclick.net	✓
United Kingdom							
10M+	com.livescore	✗	✓	✗	✗	pubads.g.doubleclick.net	-
5M+	com.genius.android	✓	✗	◖	✓	pubads.g.doubleclick.net	✓
Germany							
5M+	com.genius.android	✓	✗	◖	✓	pubads.g.doubleclick.net	✓
1M+	com.studioeleven.windfinder	✓	✗	✗	✗	googleads.g.doubleclick.net	✓
Greece							
10M+	com.ilmeteo.android.ilmeteo	✓	✗	◗	✗	pubads.g.doubleclick.net	✓
5M+	com.genius.android	✓	✗	◖	✓	pubads.g.doubleclick.net	✓
1M+	com.studioeleven.windfinder	✓	✗	✗	✗	googleads.g.doubleclick.net	✓
1M+	hurriyet.mobil.android	✓	✗	✗	✗	pubads.g.doubleclick.net	✓
1M+	com.mynet.android.mynetapp	✓	✓	✗	✗	embed.dugout.com	-
1M+	com.finallevel.radiobox	✓	✗	✗	✗	googleads.g.doubleclick.net	✓
500K+	com.famousbirthdays	✓	✗	✗	✗	pubads.g.doubleclick.net	✓
500K+	com.kupujemprodajem.android	✓	✗	✗	✗	pubads.g.doubleclick.net	✓
100K+	de.heise.android.heiseonlineapp	✓	✗	●	✗	googleads.g.doubleclick.net	✓

Browser Apps. Android besides native and hybrid apps also offers apps that are mainly used for accessing webpages. Even though their functionality is different, as they execute JavaScript code both from their native part as well as from the visited webpage, it is important to understand whether they enforce some access control policy over the privileges of in-app ads. Moreover, browser apps have the ability to display in-app ads, as well as ads originating from a website (i.e., website-ads). As such, they require manual analysis in a controlled and targeted experiment. In general, this experiment aims to identify whether in-app ads are allowed to access motion sensors and if they are displayed (or execute JavaScript) in webpages with sensitive content. Out of the most popular browser apps that are marked by Google Play as "Contain Ads", we selected those that we found that they

Table 4.4: Browsers marked by Google Play as "Contains Ads" (i.e., in-app ads) that listen to `devicemotion` and `deviceorientation` events, sorted in descending order based on the number of downloads. Intra-app vulnerability denotes that the app either displays ads in sensitive `Views` (◖) or uses Google's interstitial ad placements (◗). If both cases are found they are marked with (●). Inter-app vulnerability denotes that ads have the ability to run in the background using the app's `SYSTEM_ALERT_WINDOW` permission. Columns CAM, MIC and GPS indicate the corresponding permission.

⇓ DLs	Browser Package Names	Motion Events	Orien/tion Events	Intra-app Vuln.	Inter-app Vuln.	CAM	MIC	GPS
500M+	com.opera.mini.native	✓	✓	◖	✓	✓	✗	✓
100M+	com.opera.browser	✓	✓	◖	✓	✓	✓	✓
50M+	com.cloudmosa.puffinFree	✓	✓	●	✗	✓	✓	✓
10M+	fast.explorer.web.browser	✓	✓	●	✗	✗	✗	✓
10M+	browser4g.fast.internetwebexplorer	✓	✓	◖	✓	✓	✗	✓
10M+	com.apusapps.browser	✓	✓	◖	✓	✗	✗	✓

display in-app ads after ten minutes of interaction with them. Table 4.4 lists the browser applications that we tested for this experiment, the number of downloads and additional dangerous permissions for sensors that they hold. In order to exclude website-ads from our analysis, for each browser we visited a website with sensitive content that we know a priori that it does not display advertisements (i.e., Facebook log in page) and checked for in-app ads that are displayed on the screen and for network flows that originate from ad domains. In order to identify whether browser apps enforce some access control policy or not, for what an in-app ad (and its WebView) can access, we injected a JavaScript that accesses motion sensors only in network flows originating from advertising domains. In Table 4.4 we list the results of this experiment for browsers that fetch in-app ads from the network, after interacting with them for ten minutes. We found that none of these browsers enforce an access control policy for in-app ads that access motion sensors, and all of them allow in-app ads to capture sensor data. Even though most of the browsers tested did not display ads while visiting Facebook's log in page, we found that in-app ads displayed in the Home tab (or in any other tab) of the browser continue to access sensors even if the user switches tabs. As such, *all* browsers indirectly allow in-app ads to access sensors while a sensitive `View` is displayed, even if there is no ad in the current tab. According to Google's general policies [72] for web ads, it is forbidden to place ads in log in pages and in general this is a good security practice that should be followed by all advertisements. We observe that this is not the case with mobile apps, as Puffin Web Browser displayed a bottom in-app advertising banner in Facebook's login page. In summary, we identified that a) all browsers allow access to motion sensors by in-app ads, b) all browsers allow in-app ads to capture sensor data while a sensitive `View` is displayed, c) two browsers contains Google's interstitial ad placements and d) four browsers hold the `SYSTEM_ALERT_WINDOW` permission. We deduct that all tested browsers are vulnerable to either the intra or the inter-app data exfiltration scenario, or both.

Adult Apps. As mentioned in [43] a plethora of adult domains access motion sensors. As such, we wanted to identify whether the same behavior applies for in-app ads in

adults applications. Even though in the web counterpart there are hundreds of websites with such content, in the mobile ecosystem we could not find many adult applications (except for adult apps that act as a market place for other apps.) We performed multiple tests locally and over VPN, and we created a lot of network traffic in order to trigger as many ads as possible. Compared to advertisements displayed in other apps, results with in-app ads of adult applications did not reveal a security risk with mobile sensors since they only accessed the orientation mode of the device. Nonetheless, we found in-app ads trying to generate sound using `BaseAudioContext.createOscillator`, the `BaseAudioContext.createDynamicsCompressor` and the `OfflineAudioContext.startRendering` WebAPIs. As is the case with regular apps, there are JavaScript calls to Storage and WebGL/CanvasElements.

Chapter 5

Discussion & Limitations

In this section we discuss the emerging threat of in-app advertisements accessing rich features of the operating system and propose a set of guidelines for access control policies that should be adopted and standardized to better protect users and the ad ecosystem. Finally, we present limitations and challenging tasks that exists in any study that dynamically analyzes in-app advertisements.

5.1 Discussion

Due to the severity of the attacks enabled by mobile sensors inside in-app advertisements, it is imperative to inform the advertising community and establish guidelines for access control policies. We strongly believe that users should be given the option to allow or deny access to any sensor information. Even though access control policies enforced using Android permissions exist for sensors such as GPS, Camera, Mic, we found that it is also crucial to guard with an Android permission motion sensors. Unfortunately, even if this policy is enforced by the OS, it only partially solves the problem since in-app ads exists in the same address space as the actual application's process, and share all of the application's privileges. As such, it is also a responsibility of the World Wide Web Consortium (W3C) to update the HTML5 policies for access to motion sensors by coupling them with the Permissions API. To bridge the gap between policies of the OS and the HTML5, Android can establish a general interface that allows users to distinguish access control to sensitive data and sensors between the native part of the application and WebViews dedicated for displaying advertisements (since WebViews that are part of the core functionality of the app may require access to these sensors). These complex policies, if they are to be introduced, require careful design and a strong collaboration between OS vendors and the W3C. Bellow we list a set of guidelines that users, developers and the ad ecosystem can follow as a temporary solution until a more generic policy is enforced.

- Interstitial ads should *not* execute JavaScript before they are displayed in the screen.
- The `SYSTEM_ALERT_WINDOW` is a dangerous automatically granted permission (for apps downloaded from Google Play) and users should carefully revise which of their installed apps have been granted access.

- Apps with web content (including in-app ads) should ask for user's consent before accessing sensor information. Apps that do not require access to motion sensors for their core functionality, must *also* inform users and ask for their consent, since it is possible for embedded in-app ads to access these sensors. If user's do not agree, WebViews with in-app ads should have limited functionality (e.g., setJavaScriptEnabled(False) and display only static ads.
- Advertising entities responsible for creating, selling and publishing ads must enforce strict policies. They should not allow JavaScript in advertisements to access these sensors unless there is a specific and well documented reason to do so in the ad campaign contract. All ads must be dynamically analyzed in a sandboxed environment before publication, as to eliminate cases of suspicious obfuscated behavior.
- User's must update their OS as frequently as possible, since older Android versions have more relaxed policies concerning sensors. Security updates are mandatory and must be deployed by all device manufacturers.
- Until there is a general fix for the issues described in this paper, we urge users to use the permission option "Allow only this time" for all apps. Apps that are no longer used should be uninstalled from the device, to prevent apps from running as a background process (e.g., during device's start-up process).
- Browser apps with in-app ads should never allow them to be displayed in sensitive forms (e.g., login). Moreover, they should enforce navigational and cross tab isolation. In-app ads displayed while visiting a specific domain must not exist when visiting another domain. In-app ads displayed in inactive tabs (e.g., Home tab) should not execute JavaScript.

5.2 Limitations & Future Work

In our study we dynamically exercise multiple apps and analyze in-app ads displayed in these apps. As with any dynamic analysis study, ours also presents some limitations which we list bellow.

Coverage. To the best of our knowledge and as stated in [65], UIHarvester achieves the best coverage compared to other tools when exercising Android apps. It performs a breadth first traversal and can identify all the elements of an application. Unfortunately there are cases that any exercising tool can not cover (e.g., logical decisions). Another issue is with apps that require apriori a login. Even though we could use an SSO mechanism (e.g., Facebook's SSO), we believe that such approach may influence which in-app ads are delivered to our device. Furthermore, it is possible that we miss ad campaigns that access motion sensors because we did not execute the appropriate app in the appropriate time. Concerning VPN sessions, it is well documented that it is trivial to identify them. Although we found that in most cases using a VPN is enough for displaying foreign ads, it is possible for apps and/or ads to identify this and hide suspicious functionality or may not display ads at all. Furthermore in our analysis with VPNs, we analyzed 200 apps per session (due to VPN pricing and data limits) and the in-app ads identified over VPN sessions compared to those identified locally are fewer. In general we opted for a quantitative and qualitative

analysis, but our priority was the latter since our main goal was to understand whether in-app ads access and leak sensor data.

Interstitial ad libs. Interstitial ads are very popular and many third-party ad libs provide such functionality. We analyzed and identified vulnerabilities in Google's library, since it is one of the most popular. Reverse engineering every third-party ad lib that offers interstitial ads, requires many hours of manual analysis. Nonetheless, identifying whether other ad libraries have the same behavior with interstitial ad placements is a possible future direction.

Network flows and JavaScript. Our study involves analyzing network traffic and JavaScript for potential suspicious behavior and data leaks. The network flows and the JavaScript code analyzed in most cases was encrypted, obfuscated and also used some form of dynamic code loading to fetch additional libraries. Therefore, we manually examined such cases and included any findings in our results. Unfortunately, it is possible that we have missed cases of suspicious behavior. As such our results present a lower bound of the privacy risks posed by in-app ads that access motion sensors.

Malware. Our study identifies an emerging threat in legitimate apps from the official Google Play and advertisements shown in an appropriate ecosystem. We did not analyze malware or suspicious apps from third-party markets, therefore we do not study cases of malvertising from possible infected ecosystems.

Chapter 6

Related Work

A plethora of research studies, investigate security and privacy implications in the web and the advertising ecosystem. Our work is inspired based on several facts. Tracking users across the web has become the norm and increases revenue for advertisers by showing targeted and personalized content; mobile devices offer a plethora of features for a variety of attacks; and the extensive history with improper access control and security misconfigurations in WebViews.

HTML5 WebAPI. The standardized features of the HTML5 WebAPI allowed developers to create interactive elements, greatly improving web experience and allowing for a higher user engagement [73]. However, the rich features presented in the HTML5 WebAPI can also be misused by privacy-invasive or malicious entities. In the area of web tracking and fingerprinting the research community has extensively studied techniques that make it possible [74, 75, 76, 77, 78]. For example, Eckersley et al. [75] explored browser fingerprinting in depth and introduced the Panopticlick project for identifying common fingerprinting features in web browsers. While traditional fingerprinting techniques [79] (e.g., canvas, screen and graphics fingerprinting, etc.) are used heavily to track desktop users, smartphone devices offer additional features for this purpose. Das et al. [31] presented a study on web scripts accessing mobile sensors in 100K websites. Their study analyzes sensor-based fingerprinting by using a mobile version of OpenWPM. Contrary to privacy-invasive tracking techniques, the rich features of mobile devices can also be used for improving the users' web experience. In [80], the authors explored device fingerprinting for enhancing web authentication; while Goethem et al. [81] proposed an accelerometer-based mechanism for multi-factor mobile authentication.

Mobile Sensor Attacks Previous work showed that mobile sensors can effectively be used for device fingerprinting [15, 18, 20, 36, 16, 17, 19, 21, 22, 23] but can also be used for a plethora of side-channel attacks including speech recognition [7, 40], touchscreen input inference [25, 26, 28, 29, 44, 45, 27, 47, 46, 30, 8, 41, 82, 83, 84], location tracking [12, 85, 13, 14, 86, 37, 87, 88, 24] and physical trait or demographics inference [6, 9, 11, 89, 10, 90, 91]. Additionally, in [43] the authors provided a taxonomy of sensor-based attacks and described how they can migrate to the mobile web. Their study reveals that a majority of websites that leverage mobile-specific WebAPI calls can carry out privacy-invasive

attacks.

In-app advertising. In-app ads are an essential part of the mobile ecosystem and the defacto source of revenue for app developers. This relationship introduces several privacy issues with many personally identifiable information accessed and leaked by embedded ad libraries in mobile applications [65, 92, 93, 94]. Meng etal. [95] collected more than 200K real user profiles and found that mobile ads are personalized based on both users' demographic and interest profiles. They conclude that in-app ads can possibly leak sensitive information and ad networks' current protection mechanisms are not sufficient. Contrary to the popular belief that ad networks are responsible for user privacy, a recent study found that the privacy information presented from ad networks to developers is complying with legal regulations and app developers are the responsible entity [96]. To protect users different studies provide solutions to privacy leakage. Adsplit [97] allows the ad library to run in a separate processes with different permissions, Adroid [98] separates the privileged advertising functionality and CompARTist [99] enforces privilege separation using compiler-based instrumentation. Other more extreme solutions [100, 101, 92, 93, 102] have also been introduced that completely block advertising using network filtering or by employing a VPN approach.

WebView Risks. WebViews are essentially a mini browser embedded in many mobile apps for displaying web content. Over the years many studies showed that misconfigured hybrid applications pose a significant risk to user's privacy and Luo et al. [103] identified several attacks against WebViews. The most notorious example is the `@Javascript-Interface` that allows JavaScript code to access Java methods. In [104] the authors evaluate the impact of such possible code injection attacks using static information flow analysis, while BridgeScope [105] assesses JavaScript interfaces based on a custom flow analysis. Additionally, Mutchler et al. [106] performed a large-scale analysis of more than a million mobile apps and identified that 28% contains at least one WebView vulnerability.

Chapter 7

Conclusion

In this thesis we presented a novel attack channel, that abuses the ad ecosystem, for delivering a variety of sophisticated and stealthy attacks using mobile sensors inside in-app advertisements. Our focus is on revealing a novel attack vector that magnifies the impact and scale of such attacks, and allows attackers to stealthily reach millions of devices without the need for a malicious app to be downloaded or users to be tricked into visiting a malicious page. We analyzed the presented threat model in two distinct scenarios namely intra and inter-application data exfiltration attacks. We created a realistic dynamic framework consisting of smartphone devices that provides an in-depth view of requests to access mobile sensors and analyzed a great number of apps over a period of several months. Our findings reveal an emerging thread, since advertisements displayed inside applications across the globe access and leak motion sensor values. Apart from sensor specific WebAPIs, we found that in-app ads also access a plethora of HTML5 functions that are known to be used for fingerprinting. Due to improper access control for sensor data and the security and privacy risks they pose, we propose a set of guidelines for access control policies that should be adopted and standardized to better protect users and the ad ecosystem.

Appendices

Table 1: WebAPIs monitored by our framework.

WebAPI	Information
Mobile Specific	
Sensor APIs - Accelerometer	Provides acceleration applied to the device along all three axes.
Sensor APIs - Gyroscope	Provides the angular velocity of the device along all three axes.
Sensor APIs - AbsoluteOrientationSensor	Describes the device's physical orientation regarding Earth's reference coordinate system.
Sensor APIs - RelativeOrientationSensor	Describes the device's physical orientation without regard to the Earth's reference coordinate system.
window.addEventListener(devicemotion)	Fired at a regular interval, indicating the amount of physical force of acceleration the device is receiving.
window.addEventListener(deviceorientation)	Fired when new data is available about the current orientation (compared to the Earth's coordinate frame).
window.addEventListener(deviceorientationabsolute)	Event handler containing information about an absolute device orientation change.
window.addEventListener(deviceproximity)	Provides information about the distance of a nearby physical object.
window.addEventListener(userproximity)	Provides a rough approximation of the distance, expressed through a boolean.
window.addEventListener(devicelight)	Provides information from photo sensors or similar detectors about ambient light levels near the device.
window.addEventListener(orientationchange)	Fired when the orientation of the device has changed.
screenOrientation.addEventListener(change)	Event handler fired when the screen changes orientation.
screen.orientation.lock	Locks the orientation of the containing document to its default orientation.
screen.orientation.lockOrientation	Locks the screen into a specified orientation.
navigator.getBattery	Provides information about the system's battery.
navigator.vibrate	Pulses the vibration hardware on the device, if such hardware exists.
navigator.geolocation.watchPosition	Registers a handler function that will be called automatically each time the position of the device changes.
navigator.geolocation.getCurrentPosition	Get the current position of the device.
Generic	
XMLHttpRequest.send	The XMLHttpRequest method send() sends a request to the server
XMLHttpRequest.response	The XMLHttpRequest response property returns the response's body content
Date.prototype.getTimezoneOffset	Returns the time zone difference, in minutes, from current locale (host system settings) to UTC.
HTMLCanvasElement.toDataURL	Returns a URI containing a representation of the image in the format specified by the type parameter.
HTMLCanvasElement.getContext	Returns an object that provides methods and properties for drawing on the canvas.
WebGLRenderingContext	Interface to OpenGL ES 2.0 graphics rendering context for the drawing surface of a <canvas> element.
Storage.setItem	When passed a key name and value, will add (or update) that key to the given Storage object.
Storage.getItem	When passed a key name, will return that key's value, or null if the key does not exist.
Storage.removeItem	When passed a key name, will remove that key from the given Storage object if it exists.
Storage.key	When passed a number n, returns the name of the nth key in a given Storage object.
document.createElement(canvas)	The HTML5 <canvas> tag is used to draw graphics, on the fly, with JavaScript.
document.createElement(webgl)	A different context of <canvas> element

Bibliography

[1] H. James, "Smartphone Market Share." https://www.idc.com/promo/smartphone-market-share/os, 2020, Accessed: 2020-10-11.

[2] AdMob., "How much revenue can you earn from AdMob," https://admob.google.com/home/resources/how-much-revenue-can-you-earn-from-admob/, 2021, Accessed: 2021-02-25.

[3] M. Kaplan., "52 In-App Advertising Statistics You Should Know," https://www.inmobi.com/blog/2019/05/21/52-in-app-advertising-statistics-you-should-know, 2021, Accessed: 2021-02-25.

[4] R. Williams, "Facebook's ad revenue rises 25% to record $20.7B." https://www.mobilemarketer.com/news/facebooks-ad-revenue-rises-25-to-record-207b/571362/, 2020, Accessed: 2020-10-28.

[5] N. team, "The Nexd Perspective: Gyroscope ads engage your audience." https://www.nexd.com/blog/using-gyroscope-ads-to-better-engage-your-audience/, 2020, Accessed: 2020-10-28.

[6] E. Davarci, B. Soysal, I. Erguler, S. O. Aydin, O. Dincer, and E. Anarim, "Age group detection using smartphone motion sensors," in *Signal Processing Conference (EUSIPCO), 2017 25th European*. IEEE, 2017, pp. 2201–2205.

[7] Y. Michalevsky, D. Boneh, and G. Nakibly, "Gyrophone: Recognizing speech from gyroscope signals." in *USENIX Security Symposium*, 2014, pp. 1053–1067.

[8] T. Fiebig, J. Krissler, and R. Hänsch, "Security impact of high resolution smartphone cameras." in *WOOT*, 2014.

[9] Y. Ren, Y. Chen, M. C. Chuah, and J. Yang, "Smartphone based user verification leveraging gait recognition for mobile healthcare systems," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2013 10th Annual IEEE Communications Society Conference on*. IEEE, 2013, pp. 149–157.

[10] F. Juefei-Xu, C. Bhagavatula, A. Jaech, U. Prasad, and M. Savvides, "Gait-id on the move: Pace independent human identification using cell phone accelerometer dynamics," in *Biometrics: Theory, Applications and Systems (BTAS), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 8–15.

[11] J. Zulueta, A. Piscitello, M. Rasic, R. Easter, P. Babu, S. A. Langenecker, M. McInnis, O. Ajilore, P. C. Nelson, K. Ryan *et al.*, "Predicting mood disturbance severity with mobile phone keystroke metadata: A biaffect digital phenotyping study," *Journal of medical Internet research*, vol. 20, no. 7, 2018.

[12] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Transactions on Sensor Networks (TOSN)*, vol. 6, no. 2, p. 13, 2010.

[13] J. Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang, "Accomplice: Location inference using accelerometers on smartphones," in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*. IEEE, 2012.

[14] S. Narain, T. D. Vo-Huu, K. Block, and G. Noubir, "Inferring user routes and locations using zero-permission mobile sensors," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 397–413.

[15] A. Das, N. Borisov, and E. Chou, "Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 1, pp. 88–108, 2018.

[16] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile device identification via sensor fingerprinting," *arXiv preprint arXiv:1408.1416*, 2014.

[17] I. Amerini, P. Bestagini, L. Bondi, R. Caldelli, M. Casini, and S. Tubaro, "Robust smartphone fingerprint by mixing device sensors features for mobile strong authentication," *Electronic Imaging*, vol. 2016, no. 8, pp. 1–8, 2016.

[18] A. Das, N. Borisov, and M. Caesar, "Tracking mobile web users through motion sensors: Attacks and defenses." in *NDSS*, 2016.

[19] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "Accelprint: Imperfections of accelerometers make smartphones trackable." in *NDSS*, 2014.

[20] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "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*. ACM, 2015, pp. 191–200.

[21] A. Das, N. Borisov, and M. Caesar, "Do you hear what i hear?: Fingerprinting smart devices through embedded acoustic components," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 441–452.

[22] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device id stealthily with inaudible sound," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 429–440.

[23] I. Amerini, R. Becarelli, R. Caldelli, A. Melani, and M. Niccolai, "Smartphone fingerprinting combining features of on-board sensors," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2457–2466, 2017.

[24] J. Hua, Z. Shen, and S. Zhong, "We can track you if you take the metro: Tracking metro riders using accelerometers on smartphones," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 286–297, 2017.

[25] M. Mehrnezhad, E. Toreini, S. F. Shahandashti, and F. Hao, "Stealing pins via mobile sensors: actual risk versus user perception," *International Journal of Information Security*, vol. 17, no. 3, pp. 291–313, 2018.

[26] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." *HotSec*, vol. 11, pp. 9–9, 2011.

[27] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACCessory: password inference using accelerometers on smartphones," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM, 2012, p. 9.

[28] D. Hodges and O. Buckley, "Reconstructing what you said: Text inference using smartphone motion," *IEEE Transactions on Mobile Computing*, 2018.

[29] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, 2012.

[30] R. Spreitzer, "Pin skimming: Exploiting the ambient-light sensor in mobile devices," in *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2014, pp. 51–62.

[31] A. Das, G. Acar, N. Borisov, and A. Pradeep, "The web's sixth sense: A study of scripts accessing smartphone sensors," in *Proceedings of ACM CCS, October 2018*, 2018.

[32] F. Marcantoni, M. Diamantaris, S. Ioannidis, and J. Polakis, "A large-scale study on the risks of the html5 webapi for mobile sensor-based attacks," in *30th International World Wide Web Conference, WWW '19*. ACM, 2019.

[33] "Jmango - mobile apps vs. mobile websites: User preferences," https://jmango360.com/wiki-pages-trends/mobile-app-vs-mobile-website-statistics/.

[34] A. Developers., "App security improvement program," https://developer.android.com/google/play/asi, 2021, Accessed: 2021-02-25.

[35] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: Association for Computing Machinery, 2012. [Online]. Available: https://doi.org/10.1145/2414456.2414498

[36] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, "Deepsense: A unified deep learning framework for time-series mobile sensing data processing," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 351–360. [Online]. Available: https://doi.org/10.1145/3038912.3052577

[37] T. Watanabe, M. Akiyama, and T. Mori, "Routedetector: Sensor-based positioning system that exploits spatio-temporal regularity of human mobility," in *9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15)*, 2015.

[38] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[39] D. Genkin, M. Pattani, R. Schuster, and E. Tromer, "Synesthesia: Detecting screen content via remote acoustic side channels," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[40] S. A. Anand and N. Saxena, "Speechless: Analyzing the threat to speech privacy from smartphone motion sensors," in *2018 IEEE Symposium on Security and Privacy (SP). Vol. 00*, 2018, pp. 116–133.

[41] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "ispy: Automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 527–536. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046769

[42] J. Zhang, A. Beresford, and I. Sheret, "Sensorid: Sensor calibration fingerprinting for smartphones," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[43] M. Diamantaris, F. Marcantoni, S. Ioannidis, and J. Polakis, "The seven deadly sins of the html5 webapi: A large-scale study on the risks of mobile sensor-based attacks," *ACM Trans. Priv. Secur.*, vol. 23, no. 4, Jul. 2020. [Online]. Available: https://doi.org/10.1145/3403947

[44] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, 2012, pp. 323–336.

[45] D. Ping, X. Sun, and B. Mao, "Textlogger: Inferring longer inputs on touch screen using motion sensors," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ser. WiSec '15. New York, NY, USA: ACM, 2015, pp. 24:1–24:12. [Online]. Available: http://doi.acm.org/10.1145/2766498.2766511

[46] L. Cai and H. Chen, "On the practicality of motion based keystroke inference attack," in *International Conference on Trust and Trustworthy Computing*. Springer, 2012, pp. 273–290.

[47] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith, "Practicality of accelerometer side channels on smartphones," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012.

[48] Anssi Kostiainen, Alexander Shalamov, "Accelerometer," https://www.w3.org/TR/accelerometer/, 2018, Accessed: 2018-07-13.

[49] Rich Tibbett, Tim Volodine, Steve Block, Andrei Popescu, "Device orientation event," https://www.w3.org/TR/orientation-event/, 2018, Accessed: 2018-07-13.

[50] Mounir Lamouri, Marcos Cáceres, "Screen orientation API," https://www.w3.org/TR/screen-orientation/, 2018, Accessed: 2018-07-13.

[51] R. Waldron, "Generic Sensor API." https://www.w3.org/TR/generic-sensor/, 2019, Accessed: 2021-02-10.

[52] M. P. Alexander Shalamov, "Sensors For The Web!" https://developers.google.com/web/updates/2017/09/sensors-for-the-web, 2017, Accessed: 2021-02-10.

[53] J. Y. Mounir Lamouri, Marcos Cáceres, "Permissions." https://www.w3.org/TR/permissions/, 2020, Accessed: 2021-02-10.

[54] Google, "WebView - A View that displays web pages." https://developer.android.com/reference/android/webkit/WebView, 2020, Accessed: 2020-10-28.

[55] statcounter, "Mobile Operating System Market Share Worldwide." https://gs.statcounter.com/os-market-share/mobile/worldwide, 2021, Accessed: 2021-02-10.

[56] T. Reimers, "Axolotl: A Keylogger for iPhone and Android." https://medium.com/@tomasreimers/axolotl-a-keylogger-for-iphone-and-android-a8b7b62cdab4, 2017, Accessed: 2020-10-11.

[57] T. Reimers/Github, "Axolotl is a library that attempts to discern user screen taps from the motion of the accelerometer and gyroscope." https://github.com/tomasreimers/axolotl, 2020, Accessed: 2020-10-28.

[58] A. Developers., "Interstitial (legacy API)," https://developers.google.com/admob/android/interstitial, 2021, Accessed: 2021-02-25.

[59] A. SDK, "SYSTEM_ALERT_WINDOW permission." https://developer.android.com/reference/android/Manifest.permission#SYSTEM_ALERT_WINDOW, 2021, Accessed: 2021-02-10.

[60] Y. Fratantonio, C. Qian, S. P. Chung, and W. Lee, "Cloak and dagger: From two permissions to complete control of the ui feedback loop," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 1041–1057.

[61] Stackoverflow, "SYSTEM_ALERT_WINDOW - How to get this permission automatically on Android 6.0 and above," https://stackoverflow.com/questions/36016369/system-alert-window-how-to-get-this-permission-automatically-on-android-6-0-an, 2021, Accessed: 2021-02-10.

[62] A. SDK, "Window Layout - FLAG_SECURE." https://developer.android.com/reference/android/view/WindowManager.LayoutParams.html#FLAG_SECURE, 2021, Accessed: 2021-02-10.

[63] rovo89, "Xposed framework," https://repo.xposed.info, 2018, Accessed: 2018-06-14.

[64] P. Paganini, "Android apps use the motion sensor to evade detection and deliver Anubis malware." https://securityaffairs.co/wordpress/80037/malware/android-apps-motion-sensor.html, 2019.

[65] M. Diamantaris, E. P. Papadopoulos, E. P. Markatos, S. Ioannidis, and J. Polakis, "Reaper: Real-time app analysis for augmenting the android permission system," in *9th ACM Conference on Data and Application Security and Privacy, CODASPY '19*. ACM, 2019.

[66] Cortesi, Aldo and Hils, Mayimilian and Kriechbaumer, Thomas, "mitmproxy," https://mitmproxy.org, v. 3.0.3.

[67] Ben Alman, "Monkey-patch (hook) functions for debugging and stuff," https://github.com/cowboy/javascript-hooker, 2018, Accessed: 2018-04-23.

[68] M. Firtman, "Mobile HTML5 Compatibility on Mobile Devices," http://mobilehtml5.org/, 2018, Accessed: 2018-04-22.

[69] "Raccoon - APK downloader," 2018, https://bit.ly/1yIT4bR.

[70] P. Vallina, A. Feal, J. Gamba, N. Vallina-Rodriguez, and A. F. Anta, "Tales from the porn: A comprehensive privacy analysis of the web porn ecosystem," in *Proceedings of the 2019 Internet Measurement Conference*, 2019.

[71] Ł. Olejnik, G. Acar, C. Castelluccia, and C. Diaz, "The leaking battery," in *Data Privacy Management, and Security Assurance*. Springer, 2015, pp. 254–263.

[72] AdSense, "Ad implementation policies - Ad placement policies," https://support.google.com/adsense/answer/1346295, 2021, Accessed: 2021-02-23.

[73] A. K. Ratha, S. Sahu, and P. Meher, "Html5 in web development: A new approach," 2018.

[74] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1388–1401.

[75] P. Eckersley, "How unique is your web browser?" in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 1–18.

[76] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *Security and privacy (SP), 2013 IEEE symposium on*. IEEE, 2013, pp. 541–555.

[77] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "Fpdetective: dusting the web for fingerprinters," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 1129–1140.

[78] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, pp. 155–168.

[79] R. Upathilake, Y. Li, and A. Matrawy, "A classification of web browser fingerprinting techniques," in *New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on*. IEEE, 2015, pp. 1–5.

[80] F. Alaca and P. C. van Oorschot, "Device fingerprinting for augmenting web authentication: classification and analysis of methods," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 289–301.

[81] T. Van Goethem, W. Scheepers, D. Preuveneers, and W. Joosen, "Accelerometer-based device fingerprinting for multi-factor mobile authentication," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2016, pp. 106–121.

[82] C. Shen, S. Pei, Z. Yang, and X. Guan, "Input extraction via motion-sensor behavior analysis on smartphones," *Computers & Security*, vol. 53, pp. 143–155, 2015.

[83] S. Narain, A. Sanatinia, and G. Noubir, "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 2014, pp. 201–212.

[84] L. Simon and R. Anderson, "Pin skimmer: Inferring pins through the camera and microphone," in *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*. ACM, 2013, pp. 67–78.

[85] S. Hu, L. Su, S. Li, S. Wang, C. Pan, S. Gu, M. T. Al Amin, H. Liu, S. Nath, R. R. Choudhury, and T. F. Abdelzaher, "Experiences with enav: A low-power vehicular navigation system," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '15. New York, NY, USA: ACM, 2015, pp. 433–444. [Online]. Available: http://doi.acm.org/10.1145/2750858.2804287

[86] S. Nawaz and C. Mascolo, "Mining users' significant driving routes with low-power sensors," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '14. New York, NY, USA: ACM, 2014, pp. 236–250. [Online]. Available: http://doi.acm.org/10.1145/2668332.2668348

[87] K. A. Nguyen, R. N. Akram, K. Markantonakis, Z. Luo, and C. Watkins, "Location tracking using smartphone accelerometer and magnetometer traces," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES '19. New York, NY, USA: ACM, 2019, pp. 96:1–96:9. [Online]. Available: http://doi.acm.org/10.1145/3339252.3340518

[88] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen, "Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '15. New York, NY, USA: ACM, 2015, pp. 127–140. [Online]. Available: http://doi.acm.org/10.1145/2809695.2809718

[89] A. Bujari, B. Licar, and C. E. Palazzi, "Movement pattern recognition through smartphone's accelerometer," in *2012 IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, 2012, pp. 502–506.

[90] M. Gadaleta and M. Rossi, "Idnet: Smartphone-based gait recognition with convolutional neural networks," *Pattern Recognition*, vol. 74, pp. 25–37, 2018.

[91] N. D. Lane, P. Georgiev, and L. Qendro, "Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2015, pp. 283–294.

[92] E. P. Papadopoulos, M. Diamantaris, P. Papadopoulos, T. Petsas, S. Ioannidis, and E. P. Markatos, "The long-standing privacy debate: Mobile websites vs mobile apps," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017.

[93] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes, "Recon: Revealing and controlling pii leaks in mobile network traffic," in *MobiSys '16*, 2016.

[94] A. Continella, Y. Fratantonio, M. Lindorfer, A. Puccetti, A. Zand, C. Kruegel, and G. Vigna, "Obfuscation-resilient privacy leak detection for mobile apps through differential analysis." 2017.

[95] W. Meng, R. Ding, S. P. Chung, S. Han, and W. Lee, "The price of free: Privacy leakage in personalized mobile in-apps ads." in *NDSS*, 2016.

[96] M. Tahaei and K. Vaniea, ""developers are responsible": What ad networks tell developers about privacy," 2021.

[97] S. Shekhar, M. Dietz, and D. S. Wallach, "Adsplit: Separating smartphone advertising from applications," in *21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 553–567.

[98] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, 2012, pp. 71–72.

[99] J. Huang, O. Schranz, S. Bugiel, and M. Backes, "The art of app compartmentalization: Compiler-based library privilege separation on stock android," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1037–1049.

[100] AdGuard., "Surf the Web Ad-Free and Safely. Shield up!" https://adguard.com/en/welcome.html, 2021, Accessed: 2021-02-25.

[101] AdAway., "Ad-blocking for your Android," https://adaway.org/, 2021, Accessed: 2021-02-25.

[102] A. Shuba, A. Le, M. Gjoka, J. Varmarken, S. Langhoff, and A. Markopoulou, "Antmonitor: Network traffic monitoring and real-time prevention of privacy leaks in mobile devices," in *Proceedings of the 2015 Workshop on Wireless of the Students, by the Students, & for the Students*, 2015, pp. 25–27.

[103] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on webview in the android system," in *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011, pp. 343–352.

[104] C. Rizzo, L. Cavallaro, and J. Kinder, "Babelview: Evaluating the impact of code injection attacks in mobile webviews," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 25–46.

[105] G. Yang, A. Mendoza, J. Zhang, and G. Gu, "Precisely and scalably vetting javascript bridge in android hybrid apps," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2017, pp. 143–166.

[106] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A large-scale study of mobile web app security," in *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.