MagneticSpy: Exploiting Magnetometer in Mobile Devices for Website and Application Fingerprinting

Nikolay Matyunin
matyunin@seceng.informatik.tu-darmstadt.de
TU Darmstadt, CYSEC, Germany

Yujue Wang
yujue.wang@stud.tu-darmstadt.de
TU Darmstadt, Germany

Tolga Arul
arul@seceng.informatik.tu-darmstadt.de
TU Darmstadt, CYSEC, Germany

Kristian Kullmann
kristiandietmar.kullmann@stud.tu-darmstadt.de
TU Darmstadt, Germany

Jakub Szefer
jakub.szefer@yale.edu
Yale University, USA

Stefan Katzenbeisser
stefan.katzenbeisser@uni-passau.de
Chair of Computer Engineering,
University of Passau, Germany

ABSTRACT

Recent studies have shown that aggregate CPU usage and power consumption traces on smartphones can leak information about applications running on the system or websites visited. In response, access to such data has been blocked for mobile applications starting from Android 8. In this work, we explore a new source of side-channel leakage for this class of attacks. Our method is based on the fact that electromagnetic activity caused by mobile processors leads to noticeable disturbances in magnetic sensor measurements on mobile devices, with the amplitude being proportional to the CPU workload. Therefore, recorded sensor data can be analyzed to reveal information about ongoing activities. The attack works on a number of devices: we evaluated 80 models of modern smartphones and tablets and observed the reaction of the magnetometer to the CPU activity on 56 of them. On selected devices we were able to successfully identify which application has been opened (with up to 90% accuracy) or which web page has been loaded (up to 91% accuracy). The presented side channel poses a significant risk to end users' privacy, as the sensor data can be recorded from native apps or even from web pages without user permissions. Finally, we discuss possible countermeasures to prevent the presented information leakage.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**; **Mobile platform security**; *Browser security*; • **Hardware →** Sensor applications and deployments.

KEYWORDS

information leakage, smartphone sensors, hardware side channels, application fingerprinting, website fingerprinting, mobile security, magnetometer

WPES'19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6830-8/19/11…$15.00
https://doi.org/10.1145/3338498.3358650

ACM Reference Format:
Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. 2019. MagneticSpy: Exploiting Magnetometer in Mobile Devices for Website and Application Fingerprinting. In *18th Workshop on Privacy in the Electronic Society (WPES'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3338498.3358650

1 INTRODUCTION

Mobile devices have become ubiquitous in people's daily activities. According to recent studies, adults spend more than 2.5 hours per day on their smartphones or tablets [36], the average user runs over 30 mobile applications per month [48], while mobile Internet traffic already exceeded desktop usage [28]. Such extensive mobile usage results in an increasing amount of personal information that is stored and processed on mobile devices, which increases risks of its unauthorized or malicious misuse. Fortunately, mobile operating system developers put a great deal of effort to limit such risks, by isolating running applications into sandboxed environments and by introducing permission-based access restrictions for sensitive components [7, 18].

Nevertheless, several previous studies have shown that an attacker can exploit side-channel leakage to infer information about applications and websites opened on a victim's mobile device. These leakage sources include network traffic statistics [45, 54, 55], power consumption traces [24, 51], CPU utilization [44, 53], memory usage statistics [23, 32, 54], and other information available through the *procfs* pseudo filesystem [46] or system APIs [47, 54]. The information obtained from application and website fingerprinting can potentially reveal sensitive information about the user, e.g., hobbies, political interests, religious beliefs, or health conditions. The more actively a victim uses the device, the more precise is the resulting user profile.

To prevent such attack vectors, operating system developers have gradually restricted access to system resources which can reveal sensitive information. In particular, starting from Android 7, applications cannot access pseudofiles revealing system information about other processes (e.g., */proc/[PID]*) or monitor traffic statistics of other applications [4]. Similar access restrictions to per-process statistics are applied to applications on iOS devices starting from iOS 9 [54]. Furthermore, starting from Android 8 and on the most recent Android 9, the access to global system statistics available through *procfs* and *sysfs* is restricted [11], preventing application

and website fingerprinting attacks based on CPU utilization and power consumption traces.

In this paper, we propose an alternative source of side-channel leakage for identifying activities running on mobile devices, based on the reaction of magnetometer sensors to CPU activity. It has recently been shown that peak CPU activity on a smartphone can cause a noticeable disturbance in magnetic sensor measurements. In [37], authors utilized this observation to establish a covert channel, by encoding a payload into binary patterns of peak and idle CPU activity and analyzing the produced sensor disturbance. In this work, we propose to use this effect as a passive side-channel attack which aims to identify running activities. We show that magnetometer disturbance patterns closely represent CPU workload. Therefore, they allow to fingerprint browsing and application activity with an accuracy comparable to the method based on observing overall CPU statistics available through *procfs* before Android 8. The proposed method does not require any user permissions at the moment. As a result, any application installed on a device can infer running applications or visited websites, unnoticeable to the end user. Furthermore, the magnetometer can now be accessed within web pages using the recently-introduced Generic Sensor API [49]. In this case, the attack does not even require an installed malicious application. Instead, a web page under the attacker's control can establish fingerprinting of other web pages or applications.

We have examined 80 popular smartphones and tablets, and have found that magnetometers on 56 of them are affected by CPU activity. For these devices, we created a classifier which analyzes disturbances in recorded sensor measurements to identify activities on a device. In practical scenarios, we are able to identify an opened website with an accuracy of up to 91% for a set of 50 popular websites. We were also able to identify a running application with up to 90% accuracy for a set of 65 candidate applications. In all cases, the accuracy is significantly higher than the baseline accuracy obtained from random guessing, and is comparable to the approach based on analyzing *procfs* information. Therefore, the presented side channel can pose significant privacy risks to end users.

1.1 Contributions

Our contributions can be summarized as follows:

- We investigate the reaction of magnetic sensors to varying CPU activity on 80 different smartphones and tablets in cloud and lab environments. To the best of our knowledge, our work is the first to test this side channel on a large number of devices running both Android and iOS platforms.
- We propose to exploit this side channel for application and website fingerprinting on mobile devices. We show how to extract information from magnetometer disturbances, evaluate the classification performance under realistic conditions, and discuss possible countermeasures.
- We show that our method provides classification accuracy comparable to techniques based on *procfs* leakage, but works in presence of security enhancements implemented in the latest mobile operating systems, and can be run in both in-app (malicious app) and in-browser (malicious web page) scenarios.

2 BACKGROUND

In this section, we provide background information, describing the use of magnetometers in mobile devices and show the reaction of magnetometers to electromagnetic activity caused by the CPU.

2.1 Magnetometers

Most modern smartphones and tablets are equipped with magnetic sensors, also called magnetometers. These sensors measure the ambient geomagnetic field intensity for all three physical axes in units of micro Tesla, usually by utilizing the Hall effect [21]. Normally, they are used to estimate the orientation of the device relative to earth's magnetic north and in this way act as digital compasses, e.g., to show the user's current direction in navigation applications.

In Android and iOS native applications, 3-axis magnetometer values can be retrieved using the Sensor [14] and the Core Motion [5] frameworks, respectively. Depending on a device, the sampling rate is limited by the operating system to 50–100Hz. Access to the sensor does not require any explicit permissions, and therefore, any installed application can read sensor measurements without user attention. In web applications, magnetometer data can be accessed using the recently introduced Generic Sensor API [49], currently in W3C Working Draft status. The API is available in Google Chrome and Opera web browsers, but at the moment access to magnetometer requires a configuration flag to be explicitly enabled [42]. In comparison to native APIs, the Generic Sensor API has additional limitations: First, the data can be accessed only from the foreground tabs, and only for web pages opened using HTTPS. Second, the sampling rate is limited to 10 Hz. Nevertheless, we present attack scenarios which work even in the presence of these limitations (Section 3), while our experiments (Section 5.2) show that a sampling rate of 10 Hz does not prevent the side channel.

2.2 Sensitivity of magnetometers to CPU activity

As it has been discovered in prior work [30, 38], the magnetometer on mobile devices is susceptible to the electromagnetic radiation emanated from electronic devices located nearby. In particular, high CPU workload on a device typically requires more power, which results in a higher produced electromagnetic field. Matyunin et al. [37] showed that this effect is observable on a smartphone: Very high CPU activity on a Nexus 5X smartphone (close to 100% of the CPU load) led to a noticeable peak in magnetometer measurements.

In this work, we further investigate the reaction of magnetometers to CPU activity on mobile devices. We have observed that on many smartphones the pattern of the sensor disturbance accurately follows the pattern of the CPU utilization. The reasons for this are the following: On one hand, the CPU is one of the most power-consuming components of the device [22]. The screen and GSM module can consume more power, but their consumption remains comparably stable during normal usage. On the other hand, mobile processors are optimized to consume minimum power under low or idle activity.

At the same time, different applications or websites require different amounts of CPU resources when running. As a result, CPU utilization traces, as well as the corresponding sensor disturbance,

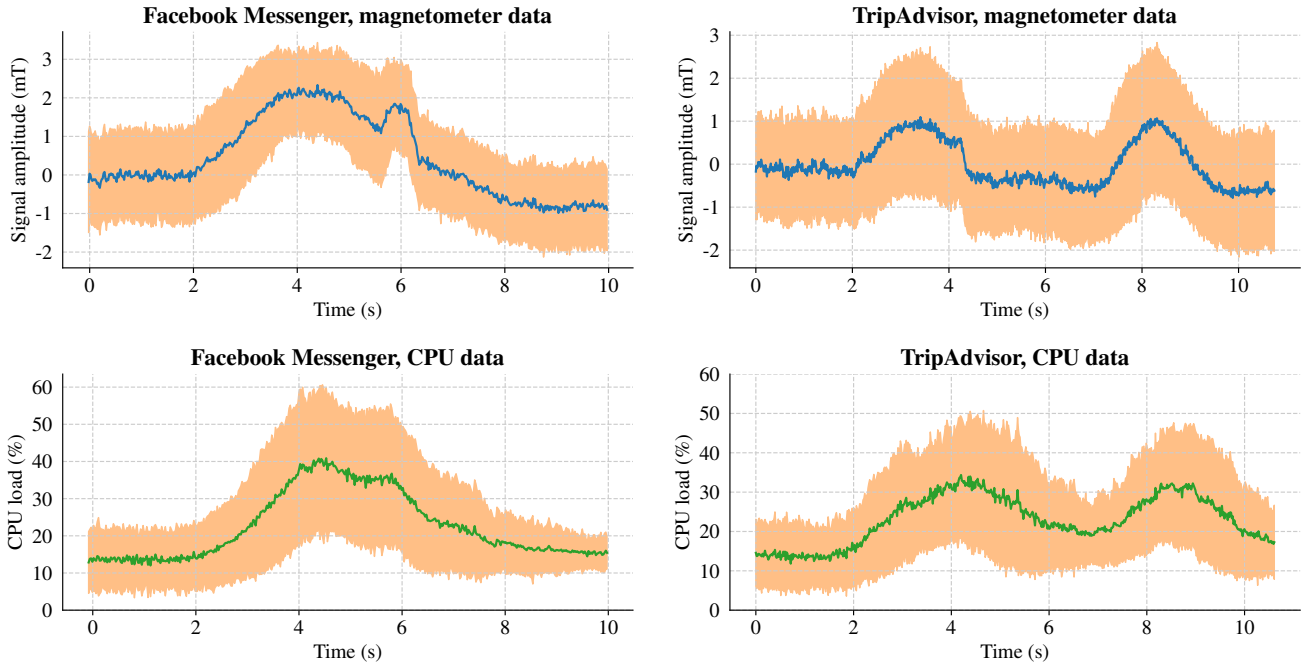

Figure 1: Examples of CPU utilization and magnetometer measurements, recorded during opening two applications on a Google Pixel 2 smartphone. Plots represent mean and standard deviation for 175 samples. The CPU and sensor data are visually correlated with each other for each application and are significantly different between applications.

can contain distinct patterns which uniquely identify the activity. Figure 1 shows CPU utilization traces recorded on a smartphone for two applications, in combination with magnetometer readings recorded at the same time. The patterns in the corresponding CPU and sensor measurements are visually correlated for each application, they are stable within multiple recordings, but distinct for two different applications. In this work, we show that an adversary can effectively extract information out of such recorded magnetometer measurements, and use it to perform application and website fingerprinting on a victim's device.

3 ATTACK SCENARIO

In this section, we discuss two considered attack scenarios, *in-app* and *in-browser*, discuss their limitations and elaborate on the considered assumptions.

In the *in-app* scenario, the victim installs an attacker-controlled application on his or her device. This application does not require privileged access rights from the system and does not have additional user permissions, apart from access to the Internet, granted by default. Therefore, malicious code can be hidden in any application which victims are likely to install. This application can be sandboxed according to the latest Android and iOS security enhancements. In particular, this application does not have any information about other running applications or network traffic, and does not have access to system resources (e.g., over *procfs* or *sysfs*). The attacker only has access to zero-permission sensor information.

In the *in-browser* scenario, a victim opens a web page under the attacker's control. The web page either fully belongs to the attacker, or contains components from an attacker-controlled server, similarly to the case when websites include third-party code from advertisement and analytics services. Such third-party components can be present on thousands of websites, which makes this scenario comparably even more scalable. Similarly, we assume that this web page is sandboxed by the browser from other web pages, processes and system resources.

In both scenarios, an attacker constantly collects magnetometer readings and tries to identify opened applications or websites by applying a supervised learning approach. To achieve this, the attacker needs to perform a training phase, which requires gathering a sufficient set of labeled traces for each visited website or application. A powerful attacker can perform learning on a large number of devices which he or she owns or accesses using cloud testing platforms, such as AWS Device Farm [8]. On a victim's device, the attacker only collects traces to be classified during the testing phase and sends them to a server. The attacker may additionally send information about the victim's device to a server, to match the victim's device with same model of the device that the attacker trained on, as model-specific classification has a higher success rate (we evaluate this in Section 5.2). On Android, the device model is freely accessible by applications through the *Build.MODEL* property; on iOS, it can be retrieved using the *UIKit* framework and *uname* system call. In the in-browser scenario, this device model can be obtained from the User-Agent HTTP header [13].

Alternatively, for the website fingerprinting case, an attacker can perform the learning phase directly on the victim's device. For this purpose, a malicious application can embed an invisible

WebView [16] component to open all websites from the training dataset and send the labeled sensor data to an attacker-controlled server. Although such an approach would provide the most precise device-specific training dataset, in this case, the application needs to be actively used in the foreground by the victim for a significant amount of time.

3.1 Applicability of the scenarios

The in-app and in-browser attack scenarios also differ regarding their applicability. Due to technical limitations of the Generic Sensor API discussed in Section 2.1, the magnetometer can only be accessed from foreground browser tabs. Therefore, the in-browser scenario can only be used to identify either *background* activities, or websites and applications opened side by side with the recording web page, in so-called *split screen* mode.

In the in-app scenario, the time frame during which the malicious application can gather magnetometer traces depends on the platform and OS version. Starting from Android 8, the background execution of applications is limited to several minutes after the last user interaction with the application [9]. In the newest Android 9, sensors cannot be accessed in the background by default [1]. To be able to continuously record sensors in the background on Android 8 and 9, the attacker needs to declare a so-called *ForegroundService* [14], which results in a visible user notification. This notification, however, can be masqueraded as a seemingly benign functionality which needs to be constantly running, e.g., a fitness activity tracker.

Similarly to Android 8 and newer, execution of iOS applications is suspended shortly after being moved to the background (starting from iOS 5). Furthermore, the iOS platform does not provide a functionality to keep the background application active for an arbitrary amount of time. However, execution time can be granted to background applications when they perform specific set of actions, e.g. playing audio, receiving location updates, processing updates from a server or reacting to remote push notifications [6].

Nevertheless, in all cases, the attacker would be able to classify applications or webpages opened shortly after the victim leaves a malicious application, classify background activities, or activity opened in split screen mode. Table 1 summarizes the differences between the in-app and in-browser scenarios.

3.2 Additional assumptions

Following other works on website and application fingerprinting, we discuss several additional assumptions [33]. In this section, we reason about these assumptions with regard to our scenarios, and show that many of them can be encountered on modern mobile platforms, in comparison to traditional desktop systems.

First, it is typically assumed that users open applications (websites) sequentially and have only a single *active* application (website) open at a time. This assumption is reasonable for our scenarios, as on mobile devices a user can not keep more than two applications in the foreground at a time, with two only in split-screen mode. Furthermore, modern mobile browsers significantly limit JavaScript execution in background tabs or even completely prevent it, to reduce power consumption. As a result, in a general case only one application remains active at a time, and, in case of a web browser, only one tab can be active.

Second, we assume that there is no user-invoked activity in the background. As described in Section 3.1, modern Android and iOS systems limit execution time of background processes. We confirmed that these limitations result in low average background activity. We performed the test measurement of the average CPU activity over a period of 24 hours on two devices with numerous applications installed and the recording application in the foreground. As a result, we obtained the average CPU utilization of only 1.9%, with the standard deviation of 1.7%. Furthermore, in the course of our experiments we did not take any measures to specifically prevent background activity. We performed our measurements on unmodified smartphones, with up to 60 additional popular applications installed. These applications could potentially generate CPU noise in the background during the continuous recording (over 30 hours of recording per device and tested scenario). Nevertheless, the high classification rates show that these activities do not significantly affect the recording traces. Overall, we can expect that the impact of background acitivtiy on the classification is low.

Third, we present evaluation results under the assumption that websites (applications) do not change over time. As observed in our experiments and other works (e.g., see [33, 52]), this assumption does not hold for websites, and the attacker needs to periodically re-run the learning phase. However, we observed that traces from applications remain stable unless they get updated. In addition, configuration options of the browser are comparably limited on mobile devices, so it is easier for the attacker to replicate the user client-side settings.

Finally, it is generally assumed that the attacker can detect the beginning and end of each activity to be classified. In practice, this can be hard to achieve: In our case, *any* CPU activity performed on a device can cause magnetic disturbances. As one potential solution, we show in Section 4.4 that the attacker can identify potential time points when the target activity could have started by computing the cross-correlation with the predefined pattern, and run the classification only at these specific points.

Apart from these assumptions, typically addressed in works on website and application fingerprinting, in this work we additionally assume that the victim is not actively moving the device, as movements affect magnetometer data. In Section 5.5, we evaluate the impact of minor movements on the classification accuracy when the smartphone is being held in hand. Furthermore, in Section 4.5, we propose an approach how the attacker can identify and filter out sensor readings which are disturbed by movements.

As a result, we believe that our scenarios are realistic under given assumptions.

4 ATTACK DETAILS

In this section, we discuss implementation details about how data was collected, its pre-processing, feature extraction and classification, describe approaches to identify the target activity in the continuous measurement stream and to identify traces disturbed by device movements.

Table 1: Comparison between the in-app and in-browser attack scenarios

	In-app scenario	In-browser scenario
Recorder	native app	web page
Sensor access	Sensor or Core Motion frameworks	Generic Sensor API
Sampling rate	50–100Hz	10Hz
Background recording	Android ≤ 7: full Android 8–9: partial (or full with notification) iOS 5 and newer: partial	n/a
Attack code distribution	Application markets	Phishing web links or 3rd-party JavaScript inclusions
Scalability	medium	high

4.1 Data collection

To collect a large set of labeled traces in the learning phase, we trigger opening applications and websites from our datasets in an automated and controllable way, using the Android Debug Bridge (*adb*) [2] tool from the Android SDK. Our service script opens each application from the dataset, waits for a predefined duration, and closes the target application. Similarly, for website fingerprinting, the service script opens the Chrome browser and the corresponding website. Additionally, we implement opening websites in a separate application with an embedded WebView component. It allows us to evaluate the website fingerprinting in the cloud testing platforms, such as the AWS Device Farm [8]. These platforms allow developers to test mobile applications remotely on multiple devices. However, they do not provide access to devices through the *adb*. Therefore, we could not evaluate the application fingerprinting or use the Chrome browser on these platforms.

To collect resulting magnetometer disturbance traces, we implemented an Android application which runs in the background, records 3-axis magnetometer data, and sends it to the attacker-controlled server. Similarly, for the in-browser scenario, we implemented a web page which records the sensors using the Generic Sensor API in the mobile Chrome browser, and sends the data to the server. As a result, for each opened application or website, the server receives labeled (in the learning phase) or unlabeled (in the testing phase) sensor measurements.

4.2 Data preprocessing

Subsequently, we convert the raw 3-axis data trace into a discrete-time one-dimensional trace. For this purpose, we apply Principal Component Analysis [31] to the data, choosing the first component as the result. The resulting data represents the one-dimensional axis with the highest data variance. Assuming that the orientation of the device is not changed significantly and that the ambient magnetic field together with EM noise is constant at a given point in time, this variance represents the vector of the EM emanation caused by the CPU. The disturbance in the one-dimensional trace can be directed above or below the baseline level. Therefore, we add both the original recorded trace and its inverse with regard to the baseline to the dataset in the training phase of the classifier, considering both traces as representations of the corresponding CPU pattern.

Finally, we normalize the result to the range [0–1], so the resulting values do not depend on the maximum possible amplitude of the disturbance (which is device-specific, see Section 5.1). Instead, the result contains information about the "shape" of the pattern, which represents the unique CPU activity pattern.

4.3 Feature extraction and data classification

Finally, we divide the resulting normalized discrete-time values of the axis with the biggest variance into equal-size overlapping intervals (bins) and calculate the mean value within each bin. These mean values are used as features for classification. To classify the traces, we use a Random Forest [20] machine learning classifier, as it outperforms other algorithms in our experiments in terms of resulting classification accuracy. We split the dataset into training set (80%) and test set (20%). The 5-fold cross-validation is performed on the training set to select optimal hyperparameters using the grid search, which include the number of estimators in the forest, the maximum number of features, maximum depth of the tree, and minimum impurity decrease. The test set was only used to compute the accuracies when evaluating the classifier in our experiments.

In our experiments, we use the RandomForestClassifier from the *scikit-learn* library [41] to perform classification. The values of the hyperparameters selected after the cross-validation are: *n_estimators* = 1100, *max_features* = *log2*, *max_depth* = 50, *min_impurity_decrease* = 0.0001. Other hyperparameters are kept as default.

4.4 Identifying target activity during continuous usage

As we discussed in Section 3.2, the attacker is assumed to know the beginning of the activity to be classified. In our case, the attacker needs to continuously monitor magnetometer disturbances, which can be caused by *any* application.

However, if the practical goal of the fingerprinting is to identify whether the victim opens a particular target application or a website (or set of or websites), we propose the following approach to reduce the amount of data to be processed by the classifier. First, the attacker can compute an averaged CPU activity pattern for the target application or website by computing mean values along multiple traces for this activity (known from the learning phase). Then, this pattern can be used to calculate the cross-correlation

with the continuously recorded data using the following formula:

$$c_{tp}[k] = \sum_{n} t[n+k]p[n],$$

where t is a recorded discrete trace and p is the computed pattern. If the target activity was produced within the recorded interval, a strong peak is present in the cross-correlation result at the corresponding time point. In practice, however, due to noise and slight changes in the produced activity patterns, cross-correlation results will not have a single strong peak, but multiple potential peaks. However, due to similarity in actual and averaged patterns, one can expect that the actual time point corresponds to one of these peaks. Therefore, the classification can be run only at time points where peaks are present in the cross-correlation result with a predefined threshold. This threshold sets a trade-off between the number of peaks and the accuracy of peak detection. We evaluate this approach in Section 5.4.

Interestingly, for website fingerprinting, an attacker can also perform this step on recorded data to first detect the web browser application to be opened (as an application fingerprinting task), and then classify the recorded interval after the browser was opened.

4.5 Identifying device movements

If the victim rotates the device, a corresponding change in the global orientation and relative direction to the magnetic north will cause a shift in magnetometer readings along three axes. In this case, the PCA-based trace will no longer represent disturbance exclusively caused by CPU activity. To identify and filter out traces which are affected by movements, we propose to analyze the rotation rate measurements from the gyroscope sensor simultaneously with the magnetometer. Access to gyroscope also does not require permissions, and its data is not affected by the CPU activity. Therefore, the attacker can use gyroscope readings to estimate if the device has been significantly moved.

More specifically, we propose two criteria for identifying traces affected by movements. The first criterion is the mean amplitude of the rotation rate along all three axes, which indicates the overall presence of movements within the recorded interval. The second criterion is the highest amplitude of the rotation rate, which indicates abrupt change in orientation. If the value computed for any of two criteria exceeds a predefined threshold, the recorded trace is considered to be affected by movements, and the trace can be ignored during the classification. In Section 5.5, we evaluate this approach for the smartphone being held in hand.

5 EVALUATION

In this section, we identify devices on which magnetometers are affected by the CPU, evaluate the classification performance, show the success rate of capturing the target activity, and investigate the impact of minor movements.

5.1 Information leakage

In this experiment, we examined whether the magnetometer readings on mobile devices are affected by the CPU workload. For this purpose, we produced a predefined CPU activity pattern on a device and analyzed resulting sensor disturbances. The pattern consists of alternating high and low CPU loads lasting for 2 seconds. To produce high loads, we concurrently ran so-called busy waiting loops in a number of threads, equal to the number of available logical cores on a device, utilizing up to 100% of the CPU time. To produce low loads, we paused the execution.

Afterwards, we calculated the correlation between this pattern and recorded measurements. If the device runs Android 7 or iOS, we were able to additionally calculate the correlation coefficient with the actual produced CPU activity pattern, recorded using */proc/stat* or *host_processor_info*, respectively. Some examples of predefined pattern and corresponding magnetometer and */proc/stat* recordings are illustrated in Figure 2. We also measured the Signal to Noise Ratio (SNR), i.e., the ratio between the average amplitude of the disturbance caused by the high CPU load and the standard deviation of measurements without CPU activity. It allows us to estimate how robust the produced disturbance is against environmental and intrinsic noise. To evaluate a large number of devices, we conducted measurements using two cloud platforms, Visual Studio App Center [15] and AWS Device Farm [8]. We selected all available devices running Android 7 or higher, and all devices running iOS 11 or higher. Additionally, five devices were used in the lab in a typical office environment. We could not control the environment of the devices in the cloud (such as noise), and tested them as is.

We found that magnetometers on 56 out of 80 devices are affected by the CPU activity. Results for selected devices are shown in Table 2, the full list is provided in Appendix A. On these devices, the disturbance clearly correlates with the CPU activity (with correlation scores over 80% on average). On most of the devices, the signal exceeds noise. In further experiments, we confirmed that a SNR of ≈4dB is sufficient to establish fingerprinting. Magnetometers on other 24 devices, listed in Appendix A, however, were not affected by CPU activity.

As one can see from both tables, the sensor model does not indicate whether the magnetometer is affected by the CPU: For example, sensors AKM AK09915 and AKM AK0991X can be found on both affected and not affected devices. We believe that the reaction mostly depends on the physical location of the sensor with regard to the CPU and power wires, and applied shielding.

As a result, we believe that the attack is practical, since modern popular devices (e.g., recently released smartphones Google Pixel 3, Samsung Galaxy S10, and iPhone XS) are all affected.

5.2 Classification results

In this experiment, we evaluated the classification accuracy of our attack in a so-called *closed-world* scenario, when the attacker aims to identify the visited website (application) among a predefined list of websites (applications).

For website fingerprinting, we collected magnetometer and CPU utilization traces during retrieval of the 50 most popular websites from the Alexa Top 500 Global Sites list [17], merging websites with multiple domains together (e.g., google.*). We collected 175 traces per website, with a duration of 12s each. Similarly, for application fingerprinting, we collected traces of 65 applications being launched, 175 traces per application, with a duration of 12s each. The applications were taken from the list of popular Android applications [3]. A full list of used websites and applications, as well as classification results for individual websites and applications,

Table 2: Selection of devices on which sensor measurements correlate with CPU activity. The full list of affected devices is presented in Appendix A. The table shows the cross-correlation between sensor data and expected CPU activity pattern (Corr.Pattern); for Android≤7 and iOS, also between sensor data and *actual* CPU loads (Corr.CPU), as well as SNR ratios.

Smartphone	Setup[a]	Magnetometer[c]	Correlation		SNR,
			Pattern	CPU[b]	dB
Android					
Google Pixel	V,A,L	AKM AK09915	0.86	0.89	14.7
Google Pixel 2	V,A,L	AKM AK09915	0.78	—	10.8
Google Pixel 3	V,A	STMicro LIS2MDL	0.90	—	14.2
Google Pixel C	V	Google CROSEC	0.91	—	27.4
Google Pixel XL	V,A	AKM AK09915	0.83	0.95	12.2
Huawei Mate 20 Pro	V	AKM	0.81	—	20.1
HTC U Ultra	V	AKM AK09915	0.95	0.96	28.6
HTC U12+	V	AKM AK09915	0.60	—	7.1
LG Nexus 5X	V,L	Bosch BMM150	0.88	0.93	15.5
LG V30	V	AKM LGE	0.93	0.96	22.6
OnePlus 3	V	MEMSIC MMC3416PJ	0.92	0.95	14.7
Samsung Galaxy Note 9	V,A	AKM AK09918C	0.52	—	4.1
Samsung Galaxy S9+	V,A	AKM AK09916C	0.55	—	4.2
Samsung Galaxy S10	V,A	AKM AK09918C	0.65	—	7.9
Sony Xperia 10 Plus	V	GlobalMEMS GMC306	0.78	—	8.8
Xiaomi Mi A1	V	AKM AK09918	0.82	—	11.0
iOS					
iPad Air 2	V,A	Unknown	0.84	0.42	13.0
iPad Mini 3	V	Unknown	0.95	0.96	16.8
iPad Pro 12.9	V,A	AKM AK8789	0.93	0.63	16.3
iPhone 5S	V,A	AKM AK8963	0.89	0.80	12.1
iPhone SE	V	Alps HSCDTD007	0.91	0.87	19.2
iPhone 6	A	AKM AK8963	0.70	0.59	8.4
iPhone 6S	V,A,L	Alps HSCDTD007	0.81	0.81	20.3
iPhone 7	V,A,L	Alps HSCDTD008A	0.89	0.85	11.0
iPhone 8 Plus	V,A	Alps e-Compass	0.87	0.81	12.0
iPhone X	V,A	Unknown	0.77	0.74	22.5
iPhone XR	V,A	Unknown	0.88	0.86	16.9
iPhone XS	V,A	Unknown	0.75	0.72	12.1

[a] V — Visual Studio App Center; A — AWS Device Farm; L — lab
[b] CPU utilization data is available only on devices running Android ≤ 7 (over */proc/stat*) and iOS (over *host_processor_info*).
[c] For Android devices, information is available from the *Sensor* API. For iOS devices, information from publicly available online resources is used.

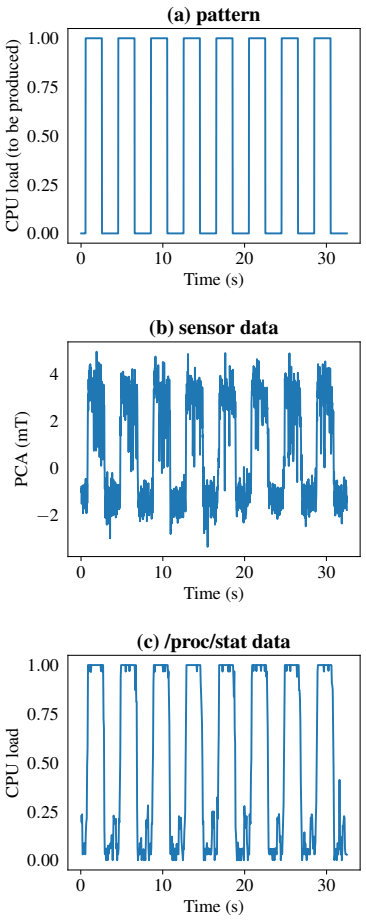

Figure 2: Example of the expected CPU pattern to be produced (a), recorded sensor data (b), and actual CPU pattern recorded through `/proc/stat` (c).

are provided in Appendix A. All traces were collected on a Google Pixel 2 smartphone lying on a table in the office environment.

Afterwards, we ran the classification using both sensor and */proc/stat* data. The results in terms of classification accuracy are shown in Table 3. As we can see, the classifier performs with an accuracy of over 80% for website and application fingerprinting. Notably, the proposed approach has a similar performance in comparison to the classification based on actual CPU activity collected through */proc/stat*. These results indicate that the magnetometer-based side channel leaks sufficient information about CPU activity.

Classification accuracies for different setups are also compared in Table 3. More specifically, we separately tested website retrieval in an embedded WebView component with cache disabled, as well as using a full mobile Chrome web browser with cache enabled. As one can see, the classification results are similar for both cases. However, the caching does affect the resulting patterns. We also achieved 86.7% accuracy with web-based recording of sensors using Generic Sensor API, which proves the applicability of our method to the in-browser scenario.

Additionally, we evaluated the classifier on a larger dataset. We increased the number of websites to 100 and repeated the experiment on a single device in the in-app scenario with an embedded WebView component. The classifier performed with an accuracy of 87.6%, comparable to 90.5% in the initial setup (see Table 3).

Table 3: Classification accuracy for website and application fingerprinting in the in-app and in-browser scenarios compared to classification using `/proc/stat` data. Traces have been collected on a Google Pixel 2 smartphone in the lab environment.

Dataset	Setup	Browser	Sampling Rate, Hz	Accuracy, %
Website fingerprinting				
sensor	in-app	Webview	100	90.5
sensor	in-app	Chrome	100	74.9
sensor	in-browser	WebView	10	86.7
cpu	in-app	Webview	50	89.0
Application fingerprinting				
sensor	in-app		100	90.0
cpu	in-app		50	95.8

Table 4: Classification accuracy for website fingerprinting in the in-app scenario for several smartphones, for intra-device and inter-device modes.

Device	Setup[a]	Accuracy, %	
		intra-device	inter-device
Google Pixel XL	V	62.5	53.2
Google Pixel 2	L	90.5	83.4
Google Pixel 3	V	83.6	80.8
HTC U12+	V	86.6	80.9
Samsung Galaxy Note 9	V	86.4	82.0
Samsung Galaxy S9+	V	81.9	78.1

[a] V — Visual Studio App Center; L — lab

Afterwards, we ran the website fingerprinting experiment on five other smartphones in the cloud environment. We calculated the success rates for intra-device (with a training and testing performed on individual devices) and inter-device (with a training phase performed on traces from all devices, and testing on individual devices) modes. The results are summarized in Table 4. Google Pixel XL and Samsung smartphones performed worse than other devices due to the lower sampling rate and the lower SNR ratio, (see Section 5.1), respectively. The activity patterns are also device-specific. Therefore, the attacker may need to train the classifier on numerous devices or take into account the target device model.

Finally, we evaluated how the sampling rate of sensor data gradually affects the classification accuracy. For this purpose, we further decreased the sampling rate for the dataset of websites recorded in the in-browser scenario and calculated the resulting classification accuracies. Figure 3 shows the results. As one can see, the sampling rate needs to be reduced to less than 1 Hz in order to make the attack impractical.

5.3 Open-world scenario

In this section, we evaluate our classifier in a so-called *open-world* scenario. In comparison to the *closed-world* scenario, (evaluated in Section 5.2), a victim can visit a much larger set of websites not known to the attacker. Consequently, the attacker cannot generalize the classifier and identify every visited website. Instead, the attacker

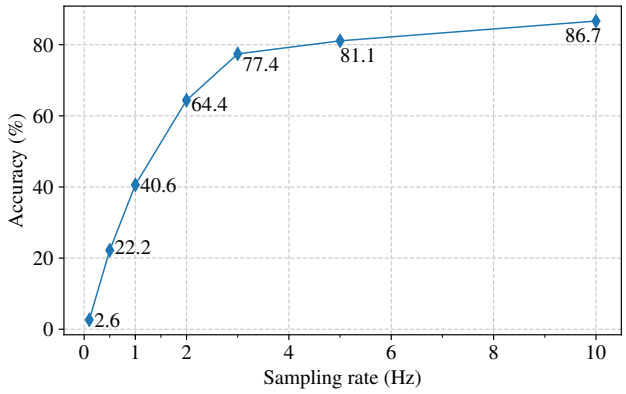

Figure 3: Classification accuracy of website fingerprinting depending on a sampling rate.

aims to identify whether a victim visits specific websites, further referred to as *monitored* websites. To perform the experiment, we first collected traces for 50 most popular websites, 175 traces per website, to train the classifier similarly to the closed-world scenario. However, in this case, we selected five the most popular websites to be five monitored classes. Other 45 websites were labeled as *not monitored*, i.e., they belonged to a separate class. For testing, we used another list of popular websites [12], which is larger than the Alexa list. We collected one trace for each of the 7,500 most popular websites, excluding 50 websites (or their alternative domains) used in the training phase. Finally, we collected 40 traces of each of the five monitored websites, to have a total of 7,700 traces in the testing set. All traces were collected on a Google Pixel 2 smartphone in the in-app recording mode.

To evaluate the results, for each class we calculated the *precision* and *recall* using the following formulae:

$$precision = \frac{TP}{TP + FP}, \; recall = \frac{TP}{TP + FN},$$

where TP (True Positives) is the number of correctly classified traces of the considered class, FP (False Positives) — the number of traces of other classes which are incorrectly classified as the considered class, FN (False Negatives) — the number of traces incorrectly classified as the other class. The previously used overall classification accuracy (99.6% in this case) is not a practical metric for the open-world experiment, as the classes are imbalanced, i.e., the number of traces for non-monitored websites significantly exceeds the number of traces for monitored websites.

Table 5 shows the classification results. We can see that the average achieved recall of 68.6% is lower in comparison to the closed-world scenario, but is still practical. The precision is, however, relatively high: 92.2% for all websites and 90.9% for monitored classes. The high precision is especially valuable in the open-world scenario, as it ensures the attacker that the victim did visit the monitored website if it was identified by the classifier. As a result, we believe that our approach is applicable to the open-world scenario.

Table 5: Classification results for the open-world scenario, in terms of the precision and the recall for the five monitored websites.

Website	Precision,%	Recall,%	F1 score,%
facebook.com	95.5	48.8	64.6
google.com	100.0	41.9	59.0
taobao.com	84.6	51.2	63.8
wikipedia.org	77.1	86.0	81.3
youtube.com	97.3	83.7	90.0
Average (monitored)	90.9	62.3	71.7
Average (overall)	92.2	68.6	76.3

5.4 Continuous usage

In this experiment, we evaluated the ability of the attacker to detect the starting point of the trace to be classified in a continuous recording stream. The detection is performed by calculating the cross-correlation with the predefined pattern, as we described in Section 4.4. We evaluated the approach in the scope of application fingerprinting and chose the Chrome browser as the target activity. We made 50 continuous recordings lasting 100s each, and within every recording we opened the target application and two other applications at specific non-overlapping time points. The applications for each recording were randomly chosen from the dataset. This way, traces contained the pattern corresponding to the target application, as well as noise from other activities. For each recording, we calculated the cross-correlation between the recorded trace and a pattern computed for the target application.

Then, we detected local maxima (peaks) in the result. The set of peaks was filtered according to three threshold parameters: peak height, prominence and width. We considered a peak as true positive if it was discovered within a 1s-interval around the time point when the target application was actually opened. Other detected peaks were considered as false positives. A false negative was assumed if there was no peak within the corresponding interval. In the end, we calculated the classification precision and recall according to these definitions, to indicate how effectively cross-classification can narrow the search area. For 50 recordings and our set of parameters, we achieved a precision of 24,5% and 72.9% recall. The attacker can vary parameters of the cross-correlation to increase the recall at the expense of precision (i.e., discover more peaks, including false positives), and vice versa.

Finally, we ran the classification at all discovered time points including false positives. This step ensures that the cross-correlation approach in the first step identifies the time points with sufficient precision, so that the classifier can correctly identify the true positive samples. The accuracy in our experiment reached 81%, which is comparable to the 90% achieved in the closed-world experiment with a known beginning point. The decrease is observed due to a number of false positives at time points corresponding to noise, as the classifier in the closed-world scenario has not being trained on noise data. As a result, the experiment shows that the attacker can efficiently reduce the amount of data to be processed using the proposed approach.

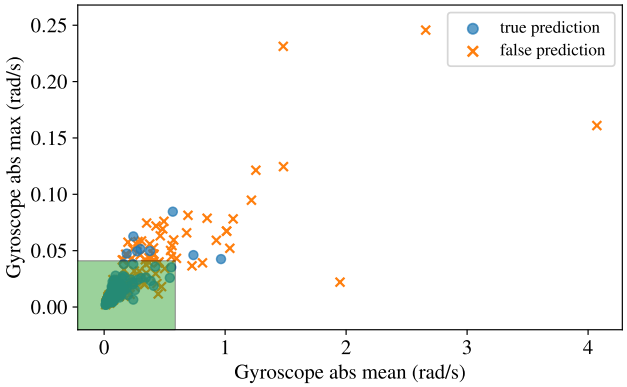

Figure 4: Distribution of traces with regard to their gyroscope-based metrics for movements. Numerous wrongly classified traces lie outside the highlighted threshold area.

5.5 Robustness to movements

In this experiment, we evaluated the classification accuracy when the smartphone is being held in hand, and our approach to identify traces affected by movements described in Section 4.5. We used the classifier trained for the website fingerprinting in the closed-world scenario on a static device (see Section 5.2). Afterwards, we recorded a total of 500 test traces while freely holding a smartphone in hand.

When the classifier was applied to the whole test dataset without filtering, the overall accuracy dropped to 64.8%, indicating that movements do affect the measurements. However, wrongly identified traces could be filtered out using the proposed approach: in Figure 4, one can see that for numerous wrongly classified traces the thresholds for indicating movements are exceeded. By applying the filtering based on the thresholds before the classification, 21% of the measurements were identified as affected by movements. After removing affected traces from the dataset, the accuracy reached 73.3%. The accuracy is lower in comparison to the accuracy achieved for the static device (90.5%), but remains practical. Nevertheless, a larger user study and more detailed analysis of the impact of movements may be needed to prove the wide applicability of the approach.

6 RELATED WORK

6.1 Website and app fingerprinting on mobile devices

Researchers have shown that different side-channel information can be used to infer applications and websites opened on a smartphone. Jana and Shmatikov [32] observed the memory footprint of a browser (available through *procfs*) to enable website fingerprinting. Zhou et al. [55] and Spreitzer et al. [45] showed that the Android data-usage statistics API provides precise information about network activity and allows to fingerprint applications and websites. Gulmezoglu et al. [29] used information about system performance counters to establish website fingerprinting, whereas Diao et al. [27]

Table 6: Comparison with other related works exploiting side-channel information leakage for website and/or application fingerprinting.

Work	Attack[a]	Leakage source	Platform	Blocked
Jana & Shmatikov [32]	WF	memory footprint	mobile apps	Android 7
Zhou et al [55]	AF	data-usage statistics	mobile apps	Android 7
Spreitzer et al. [45]	WF	data-usage statistics	mobile apps	Android 7
Gulmezoglu et al. [29]	WF	hardware perf. events	desktop apps	Android 7
Chen et al. [24]	AF	power traces (SW)	mobile apps	Android 8
Clark et al. [26]	WF	power traces (HW)	desktop apps	Android 8[b]
Yang et al. [52]	WF	power traces (HW)	mobile apps	Android 8[b]
Diao et al. [27]	AF	system interrupts	mobile apps	Android 8
Spreitzer et al. [47]	AF&WF	several Android APIs	mobile apps	not blocked
Spreitzer et al. [46]	AF&WF	several *procfs* resources	mobile apps	not blocked
Shusterman et al. [43]	WF	cache occupancy	desktop browsers[c]	not blocked[c]
Our work	AF&WF	magnetometer data	mobile apps & browsers	not blocked

[a] WF — website fingerprinting; AF — application fingerprinting
[b] Attacks use power traces collected using hardware; prevention is specified for *sysfs* traces
[c] Evaluation is presented for desktop browsers, but potentially generalizes for mobile platforms

exploited information about system interrupts to establish application fingerprinting, with both leakage sources available through *procfs*. Several researchers showed that power consumption traces, collected through *sysfs* [24, 51], using a malicious charger [51] or a malicious battery [34], are highly correlated with the CPU activity pattern, and therefore, also can be used as leakage source to infer opened applications [24, 51] and websites [26, 34, 52]. Recently, Spreitzer et al. discovered multiple leakage sources available through *procfs* [46] and Android APIs [47], which allow inferring website and application activity. Finally, several works have been presented on microarchitectural side channels, which can be used to infer information about visited websites [35, 40]. In the most recent work, Shusterman et al. [43] demonstrated the cache occupancy side channel to establish website fingerprinting in the in-browser scenario. Although the results were presented for the desktop platform, the approach may be applied to mobile devices.

Table 6 summarizes these prior works and compares them with our approach. As we can see, most of the leakage sources are already blocked in the latest Android OS. Furthermore, currently available *procfs* resources can be blocked in future versions of Android without serious impact on existing applications as they provide system-specific technical information. In contrast, our attack works on the latest Android 9 and access to magnetometer cannot be completely blocked, since numerous applications rely on magnetometer values (e.g., navigation applications). Furthermore, almost all prior works require a malicious application to be installed on a device, while our attack can be launched from a web page.

6.2 Exploiting the reaction of magnetometers to EM activity

The reaction of magnetometers to electromagnetic activity emitted by computer components has been used to establish inter-device covert channels. Researchers used magnetometers to receive covert signals from a nearby computer encoded into hard drive activity [19], CPU activity [30], and combined I/O activity [38]. Matyunin et al. [37] proposed a magnetometer-based *intra-device* covert channel on smartphones. The authors demonstrated that the magnetometer can be affected by the peak CPU activity, emitted by a webpage. In a recent work [25], Cheng et al. exploited the reaction of magnetometers to EM activity to infer applications and webpages opened on victim's laptop located in victinity to the attacker's smartphone. In this work, we show that magnetometer disturbance on mobile devices accurately represents the patterns of the *internal* CPU activity, evaluate this effect on a large number of modern devices, and show that a malevolent application on a victim's smartphone can infer running activity, namely, to perform application and website fingerprinting.

7 COUNTERMEASURES

There are several possibilities to prevent the presented information leakage through magnetometer disturbance:

- Physical shielding with ferromagnetic materials is the most straightforward way to limit the susceptibility of the sensor to electromagnetic activity. However, this measure opposes an industry trend of making smartphones thinner and lighter, and cannot protect existing devices from the attack.
- As we have discovered in our experiments in Section 5.1, some smartphones and tablets actually do not react to CPU activity, presumably due to the sensor location relative to the CPU or power supply components. We, therefore, believe that the location of the sensors should be taken into account when designing the layout of the smartphone motherboard.
- Based on our evaluation in Section 5.2, further limiting the sensor sampling rate to 1 Hz significantly reduces the classification accuracy of fingerprinting. However, with such a lower sampling rate it may be still possible to infer information about more coarse-grained activities. Furthermore, it may negatively affect the performance of legitimate applications.
- An explicit user permission can be introduced to limit access to magnetometers. However, users may not correctly perceive potential privacy threats emerging from sensors in mobile devices [39]. Therefore, an explanation of potential

risks might be needed. Moreover, a lot of mobile devices in use run outdated operating system versions [10].
- To limit the attack surface of our attack, access to magnetometers can be restricted for applications opened in the split-screen mode and can immediately be blocked when the application goes to the background.

The described countermeasures would require hardware or software changes, may have performance or production cost drawbacks, and require careful design decisions. In particular, we are concerned about the ongoing deployment of the Generic Sensor API in browsers as access to the magnetometer from the browser significantly extends the attack surface. We recommend to require an explicit permission to access the magnetometer on web pages. An alternative recommendation would be to further reduce the sampling rate.

8 DISCUSSION

In this section, we discuss some related aspects and directions for future work.

First, as shown in other works on website fingerprinting (e.g., [33, 50]), aging of sampling data affects classification accuracy. Therefore, the attacker needs to repeat the learning phase periodically. One interesting direction for future work would be a detailed investigation of which elements on a web page affect the classification the most when being changed, for different fingerprinting methods. For example, increasing the web page size by extending the text content can affect fingerprinting based on traffic analysis, but may have no substantial effect on the sensor disturbance in our approach, since text rendering is computationally inexpensive for the CPU.

Second, in principle, magnetometer sensors are suscpetible to external electromagnetic noise. However, as shown in other works [30, 38], magnetometers are affected by the noise from nearby computers only at short distances ($\leq$15cm). We performed all experiments in a typical office environment with natural arrangement of multiple electronic devices, such as laptops, WiFi access points, and other smartphones. As our results indicate, activity of these devices did not impair the performance of our approach. Nevertheless, systematic analysis of potential external noise sources and their impact on classification can be performed as future work.

Finally, it would be interesting to combine our approach with works exploiting other side-channel information on smartphones, especially leakages of other nature such as memory access statistics. In this way, a feature set combining different side-channel information can potentially improve the classification accuracy.

9 CONCLUSION

In this work, we presented a method to identify running applications and websites on mobile devices based on the reaction of magnetometers to the internal CPU activity. We observed this side channel on a large number of modern devices and demonstrated that this information leakage is sufficient to identify opened websites and applications. The presented method does not require any user permissions and can be run in both in-app and in-browser scenarios, posing a significant threat against the privacy of mobile users.

REFERENCES

[1] [n.d.]. Android 9 Behaviour changes: all apps. https://developer.android.com/about/versions/pie/android-9.0-changes-all
[2] [n.d.]. Android Debug Bridge (adb) | Android Developers. https://developer.android.com/studio/command-line/adb
[3] [n.d.]. Android market history data and ranklists | 2011 - 2018. https://www.androidrank.org/
[4] [n.d.]. Android Open Source Project: Bug 23310674: Enable hidepid=2 on proc. https://android-review.googlesource.com/c/platform/system/core/+/181345
[5] [n.d.]. Apple Developer Documentation | Core Motion Framework. https://developer.apple.com/documentation/coremotion
[6] [n.d.]. Apple Developer Documentation | Preparing Your UI to Run in the Background. https://developer.apple.com/documentation/uikit/app_and_scenes/preparing_your_ui_to_run_in_the_background
[7] [n.d.]. Application Sandbox | Android Open Source Project. https://source.android.com/security/app-sandbox
[8] [n.d.]. AWS Device Farm. http://aws.amazon.com/device-farm/
[9] [n.d.]. Background Execution Limits | Android Developers. https://developer.android.com/about/versions/oreo/background
[10] [n.d.]. Distribution dashboard | Android Developers. https://developer.android.com/about/dashboards/
[11] [n.d.]. Google Issue Tracker: 37140047: Android O prevents access to /proc/stat. https://issuetracker.google.com/issues/37140047
[12] [n.d.]. Majestic Million - Majestic. https://majestic.com/reports/majestic-million
[13] [n.d.]. MDN web docs: User-Agent - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent
[14] [n.d.]. Sensors overview | Android Developers. https://developer.android.com/guide/topics/sensors/sensors_overview
[15] [n.d.]. Visual Studio AppCenter. https://visualstudio.microsoft.com/app-center/
[16] [n.d.]. WebKit WebView | Android Developers. https://developer.android.com/reference/android/webkit/WebView
[17] Alexa Internet, Inc. [n.d.]. Alexa Top 500 Global Sites. https://www.alexa.com/topsites
[18] Apple corp. [n.d.]. iOS Security Guide. https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf
[19] Sebastian Biedermann, Stefan Katzenbeisser, and Jakub Szefer. 2015. Hard Drive Side-Channel Attacks Using Smartphone Magnetic Field Sensors. In *Financial Cryptography and Data Security - 19th International Conference, 2015*. 489–496.
[20] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
[21] Yongyao Cai, Yang Zhao, Xianfeng Ding, and James Fennelly. 2012. Magnetometer basics for mobile phone applications. *Electron. Prod.(Garden City, New York)* 54, 2 (2012).
[22] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *2010 USENIX Annual Technical Conference, Boston, MA, USA*.
[23] Qi Alfred Chen, Zhiyun Qian, and Zhuoqing Morley Mao. 2014. Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks. In *Proceedings of the 23rd USENIX Security Symposium*. 1037–1052.
[24] Yimin Chen, Xiaocong Jin, Jingchao Sun, Rui Zhang, and Yanchao Zhang. 2017. POWERFUL: Mobile app fingerprinting via power analysis. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 1–9.
[25] Yushi Cheng, Xiaoyu Ji, Wenyuan Xu, Hao Pan, Zhuangdi Zhu, Chuang-Wen You, Yi-Chao Chen, and Lili Qiu. 2019. MagAttack: Guessing Application Launching and Operation via Smartphone. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2019, Auckland, New Zealand, July 09-12, 2019*. 283–294. https://doi.org/10.1145/3321705.3329817
[26] Shane S Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. 2013. Current events: Identifying webpages by tapping the electrical outlet. In *European Symposium on Research in Computer Security*. Springer, 700–717.
[27] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. 2016. No pardon for the interruption: New inference attacks on android through interrupt timing analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 414–432.
[28] Eric Enge. [n.d.]. Mobile vs Desktop Usage in 2018: Mobile takes the lead. https://www.stonetemple.com/mobile-vs-desktop-usage-study/
[29] Berk Gülmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. 2017. PerfWeb: How to Violate Web Privacy with Hardware Performance Events. In *22nd European Symposium on Research in Computer Security, Oslo, Norway*. 80–97.
[30] Mordechai Guri, Andrey Daidakulov, and Yuval Elovici. 2018. Magneto: Covert channel between air-gapped systems and nearby smartphones via cpu-generated magnetic fields. *arXiv preprint arXiv:1802.02317* (2018).
[31] H. Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology* 24, 6 (1933), 417–441.
[32] Suman Jana and Vitaly Shmatikov. 2012. Memento: Learning Secrets from Process Footprints. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. 143–157.
[33] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014*

ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. 263–274.

[34] Pavel Lifshits, Roni Forte, Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein. 2018. Power to peep-all: Inference Attacks by Malicious Batteries on Mobile Devices. *Proceedings on Privacy Enhancing Technologies* 2018, 4 (2018), 141–158.

[35] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium*.

[36] Tina Lu. [n.d.]. Almost Half Of Smartphone Users Spend More Than 5 Hours A Day on Their Mobile Device. https://www.counterpointresearch.com/almost-half-of-smartphone-users-spend-more-than-5-hours-a-day-on-their-mobile-device/

[37] Nikolay Matyunin, Nikolaos Athanasios Anagnostopoulos, Spyros Boukoros, Markus Heinrich, André Schaller, Maksim Kolinichenko, and Stefan Katzenbeisser. 2018. Tracking Private Browsing Sessions using CPU-based Covert Channels. In *WiSec*.

[38] Nikolay Matyunin, Jakub Szefer, Sebastian Biedermann, and Stefan Katzenbeisser. 2016. Covert channels using mobile device's magnetic field sensors. In *ASP-DAC 2016, Macao*.

[39] Maryam Mehrnezhad, Ehsan Toreini, Siamak F. Shahandashti, and Feng Hao. 2018. Stealing PINs via mobile sensors: actual risk versus user perception. *Int. J. Inf. Sec.* 17, 3 (2018), 291–313.

[40] Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. 2015. The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. *ACM SIGSAC CCS* (2015). arXiv:1502.0737

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[42] Alexander Shalamov and Mikhail Pozdnyakov. [n.d.]. Sensors for the Web! https://developers.google.com/web/updates/2017/09/sensors-for-the-web

[43] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2019. Robust Website Fingerprinting Through the Cache Occupancy Channel. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 639–656.

[44] Laurent Simon, Wenduan Xu, and Ross Anderson. 2016. Don't Interrupt Me While I Type: Inferring Text Entered Through Gesture Typing on Android Keyboards. *Proceedings on Privacy Enhancing Technologies* 2016, 3 (2016), 136–154.

[45] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. 2016. Exploiting Data-Usage Statistics for Website Fingerprinting Attacks on Android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*.

[46] Raphael Spreitzer, Felix Kirchengast, Daniel Gruss, and Stefan Mangard. 2018. ProcHarvester: Fully Automated Analysis of Procfs Side-Channel Leaks on Android. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*.

[47] Raphael Spreitzer, Gerald Palfinger, and Stefan Mangard. 2018. SCAnDroid: Automated Side-Channel Analysis of Android APIs. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 224–235.

[48] Lexi Sydow and Sam Cheney. [n.d.]. 2017 Retrospective: A Monumental Year for the App Economy. https://www.appannie.com/en/insights/market-data/app-annie-2017-retrospective/

[49] Rick Waldron, Mikhail Pozdnyakov, and Alexander Shalamov. [n.d.]. Generic Sensor API. W3C Candidate Recommendation. https://www.w3.org/TR/generic-sensor/

[50] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on Tor. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013*. 201–212. https://doi.org/10.1145/2517840.2517851

[51] Lin Yan, Yao Guo, Xiangqun Chen, and Hong Mei. 2015. A Study on Power Side Channels on Mobile Devices. (2015). arXiv:arXiv:1512.07972v1

[52] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S Balagani. 2017. On Inferring Browsing Activity on Smartphones via USB Power Analysis Side-Channel. *IEEE Transactions on Information Forensics and Security* 12, 5 (2017), 1056–1066.

[53] Kehuan Zhang and XiaoFeng Wang. 2009. Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems. In *18th USENIX Security Symposium, Montreal, Canada, August 10-14, 2009, Proceedings*. 17–32.

[54] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and XiaoFeng Wang. 2018. OS-level Side Channels without Procfs: Exploring Cross-App Information Leakage on iOS. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_05B-4_Zhang_paper.pdf

[55] Xiao-yong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A. Gunter, and Klara Nahrstedt. 2013. Identity, location, disease and more: inferring your secrets from android public resources. In *ACM SIGSAC CCS*.

A APPENDIX

Table 7: Full list of mobile devices on which sensor measurements correlate with CPU activity.

Smartphone	Setup[a]	Magnetometer[c]	Correlation		SNR,
			Pattern	CPU[b]	dB
Android					
Essential Products PH-1	V	AKM AK09915	0.83	—	10.2
Google Pixel	V,A,L	AKM AK09915	0.86	0.89	14.7
Google Pixel 2	V,A,L	AKM AK09915	0.78	—	10.8
Google Pixel 3	V,A	STMicro LIS2MDL	0.90	—	14.2
Google Pixel 3 XL	V,A	STMicro LIS2MDL	0.91	—	15.4
Google Pixel C	V	Google CROSEC	0.91	—	27.4
Google Pixel XL	V,A	AKM AK09915	0.83	0.95	12.2
HTC U Ultra	V	AKM AK09915	0.95	0.96	28.6
HTC U11	V,A	AKM AK09915	0.50	—	−9.7
HTC U12+	V	AKM AK09915	0.60	—	7.1
Huawei Honor View 10	V	AKM AK09918	0.81	—	10.8
Huawei Mate 20 Pro	V	AKM	0.81	—	20.1
Huawei Mate 10 Pro	V	AKM	0.87	—	13.0
Huawei Nexus 6P	V	Bosch BMM150	0.85	0.94	14.5
Huawei P10	V	AKM	0.84	0.89	15.8
Huawei P20 Pro	V	AKM	0.60	—	3.4
Huawei P30 Pro	V	AKM AK09918	0.54	—	4.2
LG G6	V	AKM LGE	0.86	0.88	12.1
LG Nexus 5X	V,L	Bosch BMM150	0.88	0.93	15.5
LG V30	V	AKM LGE	0.93	0.96	22.6
Motorola Moto X(4)	V	MEMSIC MMC3630KJ	0.82	—	4.2
Motorola Moto g(6)	V	AKM AK09918	0.54	—	2.9
Motorola Moto Z3 Play	V	AKM AK09915	0.79	—	13.4
OnePlus 3	V	MEMSIC MMC3416PJ	0.92	0.95	14.7
OnePlus OnePlus 6T	V	AKM AK0991X	0.40	—	−0.7
Samsung Galaxy A7	V	Yamaha YAS539	0.84	—	13.8
Samsung Galaxy Note 8	V,A	AKM AK09916C	0.91	0.95	16.3
Samsung Galaxy Note 9	V,A	AKM AK09918C	0.52	—	4.1
Samsung Galaxy S7	V	Yamaha YAS537	0.73	—	4.5
Samsung Galaxy S9+	V,A	AKM AK09916C	0.55	—	4.2
Samsung Galaxy S10E	V,A	AKM AK09918C	0.77	—	12.3
Samsung Galaxy S10	V,A	AKM AK09918C	0.65	—	7.9
Samsung Galaxy XCover4	V	AKM AK09916C	0.89	0.72	−1.6
Sony Xperia XZ2	V	AKM AK0991X	0.40	—	−3.0
Sony Xperia XZ3	V	AKM AK0991X	0.49	—	1.7
Sony Xperia 10 Plus	V	GlobalMEMS GMC306	0.78	—	8.8
Xiaomi Mi A1	V	AKM AK09918	0.82	—	11.0
Xiaomi Mi A2	V	AKM AK09918	0.79	—	9.7
iOS					
iPad Pro 12.9	V,A	AKM AK8789	0.93	0.63	16.3
iPad Pro 11	V,A	AKM AK8789	0.86	0.58	12.0
iPad Air 2	V,A	Unknown	0.84	0.42	13.0
iPad Mini 3	V	Unknown	0.95	0.96	16.8
iPhone 5S	V,A	AKM AK8963	0.89	0.80	12.1
iPhone SE	V	Alps HSCDTD007	0.91	0.87	19.2
iPhone 6	A	AKM AK8963	0.70	0.59	8.4
iPhone 6S	V,A,L	Alps HSCDTD007	0.81	0.81	20.3
iPhone 6S Plus	V	Alps HSCDTD007	0.93	0.91	22.9
iPhone 7	V,A,L	Alps HSCDTD008A	0.89	0.85	11.0
iPhone 7 Plus	V,A	Alps HSCDTD008A	0.82	0.76	9.6
iPhone 8	V,A	Unknown	0.85	0.88	13.4
iPhone 8 Plus	V,A	Alps e-Compass	0.87	0.81	12.0
iPhone X	V,A	Unknown	0.77	0.74	22.5
iPhone XR	V,A	Unknown	0.88	0.86	16.9
iPhone XS	V,A	Unknown	0.75	0.72	12.1
iPhone XS Max	V,A	Unknown	0.76	0.74	11.7

[a] V — Visual Studio App Center; A — AWS Device Farm; L — lab

[b] CPU utilization data is available only on devices running Android ≤ 7 (over */proc/stat*) and iOS (over *host_processor_info*).

[c] For Android devices, information provided by the *Sensor* API. For iOS devices, information from publicly available online resources is used.

Table 8: Classification results for websites.

Website	Precision	Recall	F1 score
360.cn	0.87	0.83	0.85
accuweather.com	0.82	0.96	0.89
aliexpress.com	1.00	1.00	1.00
alipay.com	0.87	0.87	0.87
amazon.com	0.92	0.97	0.95
ampproject.org	0.86	0.90	0.88
apple.com	0.83	0.85	0.84
baidu.com	0.82	0.91	0.86
bing.com	0.88	1.00	0.93
blogspot.com	0.95	0.93	0.94
csdn.net	0.92	0.80	0.86
ebay.com	0.98	0.92	0.95
facebook.com	0.86	0.78	0.82
google.com	0.74	0.89	0.81
imdb.com	0.98	1.00	0.99
instagram.com	0.96	0.94	0.95
jd.com	0.81	0.96	0.88
linkedin.com	1.00	0.98	0.99
live.com	0.88	0.84	0.86
mail.ru	1.00	1.00	1.00
mi.com	0.98	0.82	0.89
microsoft.com	0.95	0.89	0.92
msn.com	1.00	0.90	0.95
naver.com	1.00	0.98	0.99
netflix.com	0.87	0.79	0.83
office.com	0.81	0.90	0.85
ok.ru	0.95	0.86	0.90
pinterest.com	1.00	0.91	0.96
qq.com	0.83	0.98	0.90
reddit.com	0.93	0.87	0.90
samsung.com	0.98	1.00	0.99
sina.com.cn	0.86	0.87	0.86
sm.cn	0.87	0.81	0.84
sogou.com	0.70	0.92	0.80
sohu.com	0.95	0.95	0.95
spotify.com	0.93	0.89	0.91
store.google.com	0.98	0.89	0.93
taobao.com	0.93	0.86	0.89
tmall.com	0.98	0.96	0.97
twitch.tv	0.94	0.96	0.95
twitter.com	1.00	0.98	0.99
vk.com	0.94	0.87	0.90
weibo.com	0.77	0.79	0.78
whatsapp.com	0.86	0.79	0.83
wikipedia.org	0.91	0.82	0.87
yahoo.co.jp	0.96	1.00	0.98
yahoo.com	0.87	0.97	0.92
yandex.ru	0.97	0.97	0.97
yidianzixun.com	0.89	0.89	0.89
youtube.com	0.89	0.89	0.89
Average	0.91	0.91	0.91

Table 9: Classification results for applications.

Application	Precision	Recall	F1 score
com.UCMobile.intl	0.92	0.85	0.88
com.airbnb.android	1.00	1.00	1.00
com.amazon.mShop.android.shopping	1.00	0.97	0.99
com.android.chrome	0.96	0.96	0.96
com.android.vending	0.97	0.94	0.95
com.booking	0.80	0.97	0.88
com.cleanmaster.mguard	0.91	0.94	0.92
com.cleanmaster.security	0.88	0.98	0.93
com.cmplay.tiles2	0.81	0.90	0.85
com.contextlogic.wish	0.93	0.93	0.93
com.dianxinos.dxbs	1.00	0.98	0.99
com.dianxinos.optimizer.duplay	0.83	0.98	0.90
com.dts.freefireth	1.00	0.94	0.97
com.etermax.preguntados.lite	0.91	1.00	0.95
com.facebook.katana	0.88	0.88	0.88
com.facebook.lite	0.72	0.79	0.76
com.facebook.orca	0.98	0.95	0.97
com.fingersoft.hillclimb	0.96	0.98	0.97
com.firsttouchgames.dls3	0.95	0.54	0.69
com.fungames.sniper3d	0.70	0.78	0.74
com.gameloft.android.ANMP.GloftA8HM	0.89	1.00	0.94
com.gameloft.android.ANMP.GloftDMHM	0.81	0.60	0.69
com.google.android.GoogleCamera	1.00	0.98	0.99
com.google.android.apps.maps	0.93	0.91	0.92
com.google.android.apps.photos	0.96	1.00	0.98
com.google.android.apps.translate	1.00	0.92	0.96
com.google.android.calendar	0.97	0.90	0.94
com.google.android.gm	0.92	0.96	0.94
com.google.android.googlequicksearchbox	0.91	0.93	0.92
com.google.android.play.games	0.90	1.00	0.95
com.google.android.youtube	0.88	0.95	0.91
com.imangi.templerun2	0.70	0.74	0.72
com.instagram.android	0.92	0.90	0.91
com.kiloo.subwaysurf	1.00	0.49	0.66
com.king.candycrushsaga	0.57	0.95	0.71
com.king.farmheroessaga	0.98	0.98	0.98
com.lenovo.anyshare.gps	0.97	0.95	0.96
com.linkedin.android	0.85	0.95	0.90
com.miniclip.eightballpool	0.91	0.96	0.93
com.mobile.legends	0.93	1.00	0.96
com.mxtech.videoplayer.ad	0.96	0.84	0.90
com.nekki.shadowfight	1.00	0.93	0.96
com.ngame.allstar.eu	1.00	0.98	0.99
com.outfit7.mytalkingangelafree	0.88	0.70	0.78
com.outfit7.mytalkingtomfree	0.90	0.91	0.91
com.paypal.android.p2pmobile	0.98	1.00	0.99
com.picsart.studio	0.85	0.83	0.84
com.qihoo.security	0.91	0.98	0.94
com.quvideo.xiaoying	0.98	0.94	0.96
com.roidapp.photogrid	0.94	0.96	0.95
com.skype.raider	1.00	0.85	0.92
com.snapchat.android	0.98	1.00	0.99
com.spotify.music	0.92	0.90	0.91
com.supercell.clashofclans	0.70	0.82	0.75
com.supercell.clashroyale	0.98	0.96	0.97
com.supercell.hayday	0.78	0.90	0.84
com.surpax.ledflashlight.panel	0.97	0.66	0.79
com.tencent.ig	0.92	0.58	0.71
com.tencent.mm	0.92	1.00	0.96
com.tripadvisor.tripadvisor	1.00	1.00	1.00
com.twitter.android	0.95	0.74	0.83
com.waze	0.89	0.98	0.93
com.whatsapp	0.91	0.94	0.93
com.zhiliaoapp.musically	0.73	0.92	0.81
jp.naver.line.android	1.00	0.92	0.96
me.pou.app	0.84	0.97	0.90
net.zedge.android	0.65	0.82	0.73
Average	0.90	0.90	0.90

Table 10: List of mobile devices and their magnetometers not affected by their CPU activity.

Smartphone	Setup	Magnetometer
Android		
Asus ZenFone 5Z	V	AKM AK0991X
Google Pixel 2 XL	V,A	AKM AK09915
LGE LG G7 ThinQ	V,A	AKM LGE
Motorola Nexus 6	V,A	Invensense Inc.
Motorola Moto G(6) plus	V	AKM AK09918
Motorola Moto G(7) plus	V	MEMSIC MMC5603NJ
Motorola One	V	MEMSIC MMC3630KJ
OnePlus 5T	V	AKM AK09911
OnePlus 6	V	AKM AK0991X
Samsung Galaxy A6+	V	Yamaha YAS539
Samsung Galaxy A8	V	AKM AK09918
Samsung Galaxy S7 edge	V	Yamaha YAS537
Samsung Galaxy S8	V,A	AKM AK09916C
Samsung Galaxy S8+	V,A	AKM AK09916C
Samsung Galaxy S9	V,A	AKM AK09916C
Samsung Galaxy S10+	V,A	AKM AK09918C
Samsung Galaxy Tab S2	V	Yamaha
Samsung Galaxy Tab S3	V,A	AKM AK09916
Samsung Galaxy Tab S4	V	AKM AK09918
Sony Xperia 10 Plus	V	GlobalMEMS GMC306
Sony Xperia XZ2 Compact	V	AKM AK0991X
iOS		
iPad Air 2019	V	Unknown
iPad Mini 4	V	Unknown
iPad Pro	V	AKM AK8789
iPad Pro 9.7	V	Unknown