1-2020

DeepMag+ : Sniffing Mobile Apps in Magnetic Field Through Deep Learning

Rui Ning

Cong Wang

ChunSheng Xin

Jiang Li

Hongyi Wu

Pervasive and Mobile Computing 61 (2020) 101106

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

DeepMag+: Sniffing mobile apps in magnetic field through deep learning

Rui Ning *, Cong Wang, ChunSheng Xin, Jiang Li, Hongyi Wu

Old Dominion University, Norfolk, VA 23529, USA

ARTICLE INFO

Article history:
Received 10 July 2018
Received in revised form 14 September 2019
Accepted 4 October 2019
Available online 26 November 2019

Keywords:
Deep learning
Privacy
Magnetic
Smartphone

ABSTRACT

This paper reports a new side-channel attack to smartphones using the unrestricted magnetic sensor data. We demonstrate that attackers can effectively infer the Apps being used on a smartphone with an accuracy of over 80%, through training a deep Convolutional Neural Networks (CNN). Various signal processing strategies have been studied for feature extractions, including a tempogram based scheme. Moreover, by further exploiting the unrestricted motion sensor to cluster magnetometer data, the sniffing accuracy can increase to as high as 98%. To mitigate such attacks, we propose a noise injection scheme that can effectively reduce the App sniffing accuracy to only 15% and at the same time has a negligible effect on benign Apps.

1. Introduction

Smartphones have become constant companions in our daily life. People are not just using their mobile devices at work and home — they are living on them. People rely on smartphone applications (Apps) for communication, social networking, entertainment, banking, shopping, navigation, healthcare, and more. For many people, every day begins and ends with checking their smartphones. As more and more personal data are stored on and processed and transmitted by smartphones, they are becoming an increasingly attractive target for cybercriminals. For example, recent studies have shown several attacks by exploiting smartphone sensors [1–4].

The modern smartphones integrate a diversity of sensors for productivity, such as accelerometer, gyroscope, magnetometer, GPS, gravity sensor, barometer, microphone, ambient light sensor, and proximity sensor. The smartphone operating systems often create their own access control for the sensors. Some sensors, for instance, the location sensor (GPS) and audio sensor (microphone) trigger privacy concerns and hence require user permission before any App can access their data. On the other hand, a range of sensors (including accelerometer, gyroscope, and magnetometer), are called unsupervised sensors, which are not considered explicitly sensitive and thus accessible freely by Apps. Some of them can be even obtained by Javascript embedded in webpages [3].

In this paper we report a newfound vulnerability on smartphones due to the malicious use of unsupervised sensor data, where the attacker can sniff mobile Apps, i.e., infer what Apps have been installed on the user's mobile device, how frequently they are used, and when they are opened, by analyzing the magnetometer data along with the motion sensor data using deep learning techniques. The accuracy can be as high as 98%.

* Corresponding author.
E-mail addresses: rning001@odu.edu (R. Ning), c1wang@odu.edu (C. Wang), cxin@odu.edu (C. Xin), JLi@odu.edu (J. Li), h1wu@odu.edu (H. Wu).

https://doi.org/10.1016/j.pmcj.2019.101106
1574-1192/

1.1. Related work

1.1.1. APP fingerprinting

A range of works have attempted to infer app usage on smartphones. Various methods have been leveraged, for instance, power usage [5], system behavior [6], and network traffic [7,8]. In [5,6], the authors developed malicious Android Apps to collect system information, such as current, voltage, network state, CPU and memory usage, from a victim's device. The collected data were analyzed and machine learning techniques were used to identify the Apps on the victim's device. In [7,8], the authors designed learning systems to automatically fingerprint an App using the encrypted network traffic (e.g., IP, protocol type, length, etc.).

Compared with their approaches, our work does not require to maliciously deploy an additional App [5,6] or a traffic sniff device [7,8] to collect data on the victim's device. Therefore, our approach can launch the sniffing attack easily by incentivizing users to visit our website. It is also worth to mention that their approaches perform poorly in scenarios that the Apps have very limited network traffic, or have random system behavior caused by multithreading. In contrast, our approach is still effective under those scenarios since they do not affect magnetic sensor reading of smartphone.

1.1.2. Sensors

Recent studies introduced several attacks due to the malicious use of magnetometer and motion sensor. For instance, a range of works [1,2,5] raised the idea of cyber-vulnerability in 3D printing where a piece of malicious software can infer the design files by sniffing magnetometer. Furthermore, two recent work [3,4] proposed to utilize motion sensors in smart devices to infer a user's typed words or passwords. Similarly, Narain et al. [9,10] demonstrated that the accelerometer and the gyroscope could be used to infer car driving routes and fingerprint devices. However, none of those works bridges the correlation between magnetometer and LED screen as by our study.

1.2. Preliminary experiments and observations

The quest begins with the observation of a subtle correlation between an LED display and its surrounding magnetic field. For example, in our first experiment, we display a black image on a 27 inch LED PC monitor for 20 s, followed by a white image for 60 s. We repeat the pattern for a number of rounds. In the meantime, an iPhone 7 Plus is placed in front of the monitor about 10 cm away to measure the magnetic field. We observe a noticeable change in the magnetic field while switching between the two images (see Fig. 1(a)).

The above phenomenon motivates us to further explore how the LED display on a smartphone would affect its magnetometer readings. To this end, we carry out a similar experiment by displaying the black and white images on the iPhone 7 Plus while recording the data captured by the magnetometer on the same phone. Compared with the previous setting, we expect more stable results since the distance between the LED display and the magnetometer is shorter and their relative orientation is fixed. The experimental data are depicted in Fig. 1(b), demonstrating significant changes in magnetic field when different images are displayed on the phone.

Fig. 1(c) further shows the change of magnetometer readings while the smartphone displays four different colors, white–black–red–blue, in sequence. While it is beyond the scope of the paper to fully model this physical phenomenon, it is largely due to the fact that different bias voltages are used when LED displays different colors, which accordingly lead to the change of the magnetic field [11].

The results shown in Fig. 1 demonstrate the correlation between colors on LED display and surrounding magnetic field. Since different Apps often adopt different graphic designs that mix different color patterns, we speculate that they also induce different magnetometer readings. In particular, when one clicks on an App's icon, a unique welcome-page will be displayed till the App is fully open. The corresponding changes in magnetic field can be measured by the integrated magnetometer. Fig. 2 illustrates the averaged magnetometer readings over the period for opening two popular Apps, i.e., Snapchat and Twitter, on an iPhone 7 Plus. As can be seen, they differ dramatically on at least one axis (in this case, Y-axis). This is because Snapchat adopts predominantly a bright yellow color while Twitter uses blue. We have verified that the above observations are repeatable on different smartphones and models. More results will be presented in Sections 2–4. These preliminary experimental data indicate that different Apps are likely associated with unique magnetic signatures. Therefore, if one can acquire magnetometer data, he can potentially infer the Apps running on a smartphone.

1.3. Our contributions

This is the first work that discovers and reports the correlation between magnetometer readings and LED displays on smartphones. Based on this observation, we demonstrate a newfound side-channel attack, where the attacker can sniff mobile Apps through magnetometer data. In particular, we devise deep Convolutional Neural Networks (CNN) that can be trained to effectively infer the Apps installed on the smartphone and their usage information. We implement the attack on iPhone 7 Plus and Samsung Galaxy 7 and carry out extensive experiments, showing that the attack can achieve an accuracy of 80%–90% based on the magnetometer data.

Furthermore, we discover that the orientation data is closely correlated with the magnetometer readings. To this end, we use the orientation of the smartphone as an alternative to train and test the CNN models. The performance is only

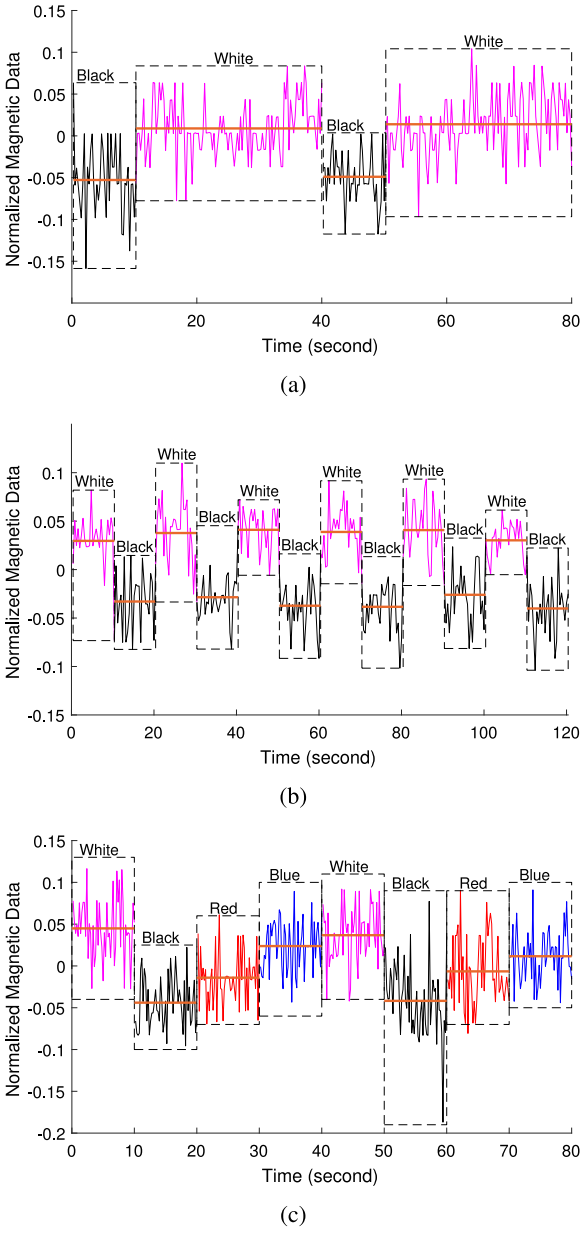
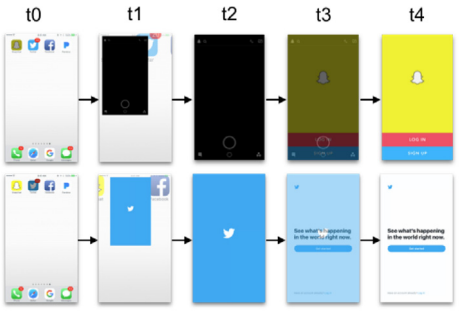


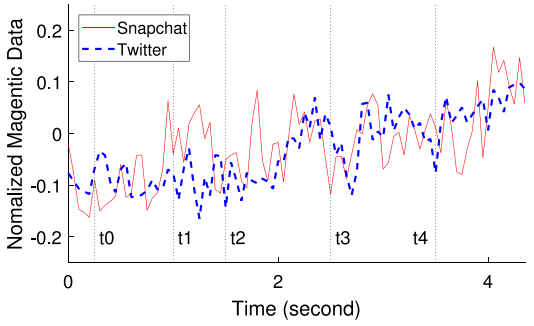
Fig. 1. Magnetometer readings on Y-axis. (a) Change of magnetometer reading due to PC LED display, where the black and magenta waveforms correspond to the black and white images, respectively. (b) Change of magnetic field due to smartphone LED display (black and white). (c) Change of magnetic field while the smartphone sequentially displays white–black–red–blue. The solid orange line in each dashed rectangle box represents the mean of the signal within the box.

slightly lower than the original approach based on magnetometer data. This finding enables even more pervasive attacks since the orientation data can be readily obtained by integrating a 4-line Javascript code in attacker's websites and the Javascript can continuously acquire the orientation data even in the background.

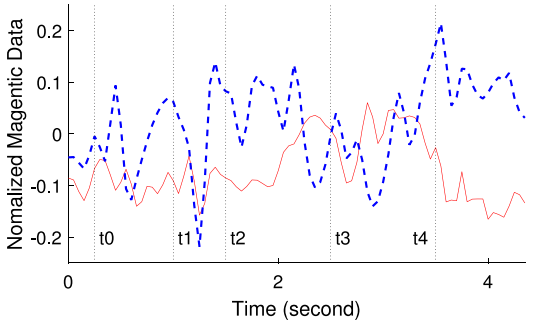
Moreover, we show that the attack can become worse if the attacker exploits motion sensor data. Briefly, for a given mobile phone, the locations of the Apps are generally fixed on the screen. Suppose the mobile user clicked on an App for k times during a period. If the k magnetometer or orientation data are fed into the CNN model, over 10% of them would be misclassified to other Apps. However, if the attacker is able to recognize that the k clicks are all at the same location on the screen, then he can put them into a cluster and feed the cluster of magnetometer or orientation data to the CNN model. The vast majority of them (about 80–90%) should be recognized correctly. Since the cluster of clicks are from the same location, they should be the same App. Therefore, a majority vote can effectively determine the App for the entire cluster. This Approach can increase the accuracy to as high as 98%.



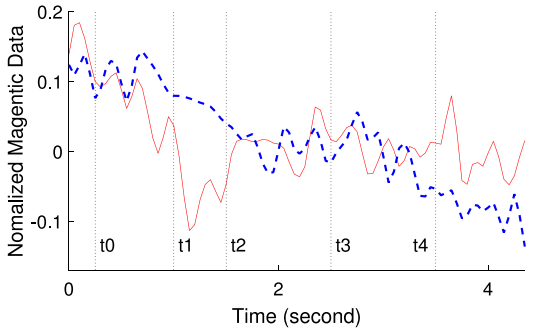
(a) Screen shots when opening Snapchat and Twitter.



(b) Magnetometer X-axis readings for Snapchat and Twitter.



(c) Magnetometer Y-axis readings for Snapchat and Twitter.



(d) Magnetometer Z-axis readings for Snapchat and Twitter.

Fig. 2. Welcome pages and magnetometer readings of popular Apps.

Besides showing this newfound side-channel attack, we also discuss viable methods to mitigate it by injecting minimal amount of noise into the magnetometer or orientation data. The rest of this paper is organized as follows. Section 2 introduces the magnetometer-based App sniffing, including extended discussions on attacks based on orientation data.

Section 3 demonstrates the motion sensor-assisted approaches. Section 4 presents experiments and results. Section 5 discusses viable schemes to defeat the attacks. Finally, Section 6 concludes the paper.

2. Magnetometer-based App sniffing (MAS)

In this section, we present App sniffing solely based on the readings from magnetometer. The preliminary observations presented in Section 1 indicate that different Apps are likely to induce different magnetic field. However, it remains a nontrivial problem to identify the unique magnetic signature for each App and accordingly infer the Apps according to magnetometer readings. The fundamental challenge lies in the facts that the magnetometer data exhibit noise and randomness and that the Apps' graphic designs often incorporate complex combination of color patterns, rendering simple classification methods infeasible. To this end, we propose to exploit the powerful deep *CNN* to classify magnetometer data and to infer the corresponding Apps.

2.1. Deep CNN models for magnetometer-based App sniffing

Magnetometer data can be recorded when a user opens an App. In our preliminary exploration, we have considered the top-7 most used Apps, i.e., Twitter, Snapchat, Pandora, Netflix, Google Maps, Chase Bank, and HBO. The recording process essentially collects a sequence of magnetometer samples during the interval from clicking on an App to the time when the welcome page of the App is fully displayed on the mobile screen. Fig. 2 shows the examples for opening Snapchat and Twitter, respectively. Different phones may have different sampling rates of their magnetometers. For instance, Samsung Galaxy S7 and iPhone 7 Plus sample their magnetometers at the rate of 20 Hz and 100 Hz, respectively. As to be shown later, the sampling rate has negligible impact on the effectiveness of App sniffing.

Since the change of magnetic field is relatively small, we preprocess the raw magnetometer data by using a de-noising function (e.g., *wden* available in MATLAB [12]). It decomposes the signal into wavelets and performs thresholding on wavelets coefficients.

We set the function at the denoise-level 5 with soft thresholding rule for the universal threshold $\sqrt{2\ln(\cdot)}$. Then, we normalize the de-noised data by subtracting its mean and dividing it by its vector's norm. Figs. 2(b)–(d) illustrate the normalized magnetometer readings of Snapchat and Twitter on X, Y, and Z axis. While the figure only shows about 4 s, the total recording time is 8.25 s including some overhead before and after the App is opened.

Thus on an iPhone with a sampling rate of 100 Hz, each magnetometer data would have a dimension of 825×3. Directly feeding such high dimensional data into the CNN yields poor results (less than 50% accuracy) because drastic changes over such a large span of 825 sample points may easily overwhelm a neural net's representational capability. To this end, we adopt a sliding window approach with a window size of W sample points to slide over the time series. Each pair of adjacent slices has an overlap of P sample points. For example, assume $W = 36$ and $P = 31$. If we have 100 original data (each with a dimension of 825×3), they will be converted to 15,800 sliced data each with a dimension of 36×3. The sliced data are labeled with corresponding Apps for training and testing as to be discussed next.

The success of CNN has been demonstrated in computer vision [13–15]. Compared to traditional learning techniques based on hand-crafted features, CNN can be trained from end-to-end to extract features automatically. It usually stacks multiple convolutional layers, each comprising filters to capture spatial patterns in the data and activations that represent the result of convolution. CNN exploits these layered structures of non-linearities to characterize highly complicated relations in the data. Max pooling layers are usually introduced between convolutional layers to reduce the number of parameters and speed up computation.

We have explored several deep CNN architectures to sniff Apps based on magnetometer data. In contrast to computer vision (that adopts 2D filters for images), the magnetometer data on smart phone involves 3 channels (x, y, z) and data points along each channel dimension is a 1D time series. To this end, 1D filter is adopted in the architectures to capture temporal correlations on each channel. Similar approaches are also found in human activity recognition [16,17].

Our goal is to demonstrate that an appropriately designed CNN can effectively sniff frequently used Apps on a mobile phone. The exploration begins with a 3-Layer architecture consisting of only one convolutional layer. Based on our observation, the variation of the Magnetic signal caused by the LED correlation is relatively small. To extract such a minor change, we select kernel size as 1×3. Each layer applies *Rectified Linear Units* (ReLUs) as the activation functions (taking $f(x) = \max(0, x)$). The features extracted by convolutional layers are fed into a densely connected layer which connects to the output softmax function.

Although a single convolutional layer is fast for computation, low-level features captured in the first layer may not generalize well on the entire dataset. To exploit the wealth of data that an attacker can obtain, we have further investigated several deeper structures by stacking more convolutional layers together and inserting max pooling layers to reduce dimensionality. Fig. 3 illustrates an example of such deep CNN structures. For brevity, a 6-layer CNN is denoted as: *Conv*(64)-*Conv*(64)-*Pool*-*Conv*(64)-*Conv*(64)-*Pool*-*Dense*(128)-*Sfmax*. A layer is counted if it has adjustable weighted connections. Each convolutional layer has 64 filters and the densely connected layer has 128 neurons with ReLU activations. Maxpooling layer reduces the input dimension by half. As more convolutional layers are stacked up, the network will be able to extract high-level features and generalize on the dataset.

To understand the effectiveness of this CNN approach and compare the accuracy of different CNN architectures, we have carried out a set of preliminary experiments. We have considered the top-7 most used Apps as discussed earlier

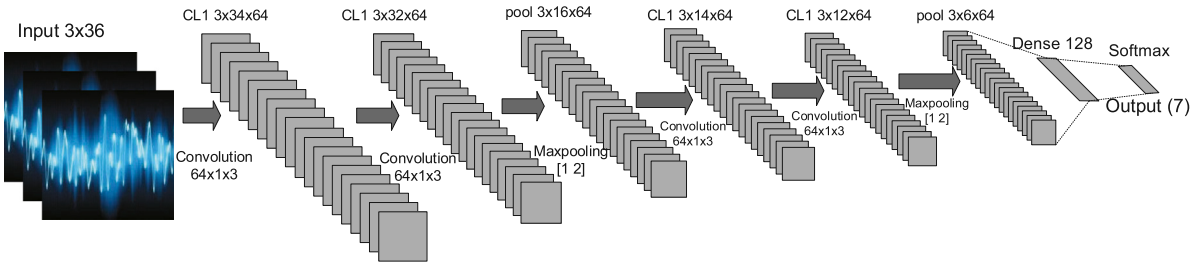
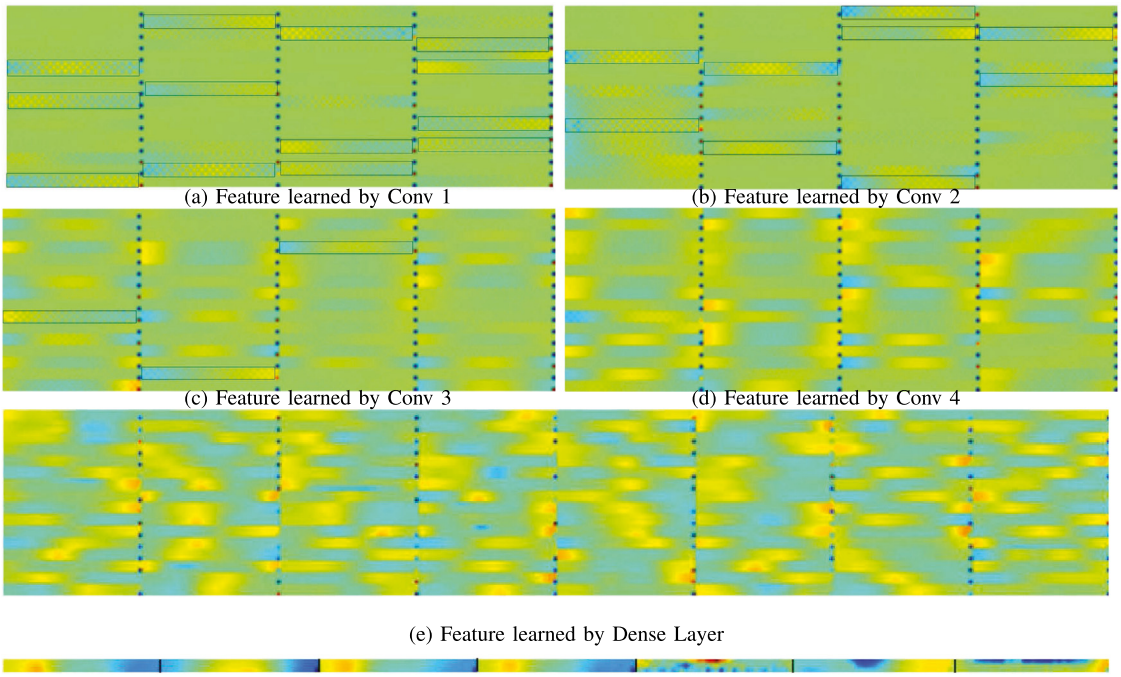


Fig. 3. An example of the proposed deep CNN architectures.



(a) Feature learned by Conv 1

(b) Feature learned by Conv 2

(c) Feature learned by Conv 3

(d) Feature learned by Conv 4

(e) Feature learned by Dense Layer

(f) Feature learned at Output Layer (Prediction)

Fig. 4. Visualization of features learned by the deep CNN.

(from Twitter to HBO), and collected a total of 700 raw magnetometer recordings on a number of Samsung Galaxy S7 and iPhone 7 Plus units. They are the flagship smartphones of Samsung and Apple — two companies that together have a total market share of 72.8% in the US [18]. The CNN models have been implemented in *Tensorflow* [19] with batch sizes of 150 and 100 epochs. For comparison, we have also implemented a baseline 3-layer neural network (NN) model that has dense connections and a support vector machine (SVM) model using LibSVM [20]. All of them are trained and tested on a PC with I7-4470 CPU and NVIDIA 1080Ti graphic card. As discussed earlier, we adopt a sliding window approach to slice each recorded magnetometer data. The default parameters are $W = 36$ and $P = 31$.

The primary performance metric is the accuracy, i.e., the fraction of correctly recognized Apps. We utilize 4-fold cross validation (CV) for performance evaluation, where we randomly divide a dataset to 4 parts and use three parts for training and the remaining part for testing. This process is repeated four times such that each part is used for testing once. We are also interested in the running time. For a 6-layer CNN model, the training time for each epoch is around 4 s, and it takes about 100 epochs to achieve converged result.

As shown in Table 1, the 6-Layer CNN has the highest accuracy. At the same time, we also observe that the accuracy is not sensitive to different CNN architectures. All of them perform significantly better than SVM and NN. Thus, an attacker can utilize any general-purpose CNN to construct the attack without the need to fine-tune the CNN model.

Fig. 4 visualizes the features learned on input signal captured by the 4 convolutional layers, dense layer and finally the output layer. Each convolutional layer trains a number of filters to match similar spatial patterns in the input signal in order to minimize the cost function. Each learned feature is displayed as 3×36 (size of the signal input) and arranged in a stitched $16 \times 4 = 64$ array for each layer. Here, 3 is the number of channels and 36 is the sliding window size. Since

Table 1
Performance comparison of different CNN models.

Machine Learning Model	Accuracy
SVM	39%
SVM with sliced input data	42%
3-Layer NN	43%
3-Layer NN with sliced input data	46%
3-Layer CNN with sliced input data	82%
5-Layer CNN with sliced input data	83%
6-Layer CNN with sliced input data	**83%**
7-Layer CNN with sliced input data	83%
8-Layer CNN with sliced input data	83%

Table 2
Confusion matrix (under 6-layer CNN with sliced input).

	HBO	Chase	Google Maps	Netflix	Pandora	Snapchat	Twitter
HBO	70.13%	1.71%	0.27%	14.92%	2.85%	1.34%	1.41%
Chase	3.78%	85.54%	3.26%	1.09%	1.12%	2.26%	2.32%
Google Maps	3.33%	2.1%	86.53%	1.46%	1.23%	1.34%	1.43%
Netflix	16.56%	1.26%	3.64%	72.79%	1.48%	1%	0.37%
Pandora	0.7%	3.52%	0.68%	2.45%	89.12%	1.19%	3.02%
Snapchat	3.14%	2.24%	1.17%	2.94%	2.86%	91.81%	0.12%
Twitter	2.36%	3.63%	4.45%	4.35%	1.34%	1.06%	91.33%

the features learned from the signals are rather flat, we remove the margins between neighboring features for better visualization. From Fig. 4(a), the highlighted features (boxed) show mosaic patterns that are activated from the raw input signal. Subsequent layers are more abstract and such low-level, mosaic patterns start to disappear from the 3rd layer. Although the high level features learned by the CNN are not visually explainable at this point [21], they jointly represent unique identifications for each class of Apps on the high level. The learned features from convolutional layers are fed into the dense layer that classifies the outputs into 7 classes. The output layer is a generalization of all the features learned by the network that average over the data points in each class. This is consistent with the reasoning in [22]. In contrast, SVM and NN fail to effectively identify different Apps. It may be due to the fact that the time series from different Apps are overlapped and these models lack the automatic feature learning capabilities.

We further compare the performances by tuning different hyperparameters. The experimental results (omitted here due to space limit) show that the model is not sensitive to the batch size. The slice window with $W = 36$ and $P = 31$ always yields the highest accuracy. We also observe that the errors are generally evenly distributed unless two Apps are very similar in colors and patterns. As shown in Table 2, the confusion between HBO and Netflix is relatively high, i.e., most errors of Netflix are misclassified into HBO, and vice versa, because their colors are both predominantly black. This is also reflected in Fig. 4(f), where the 1st and 4th classes, which respectively correspond to HBO and Netflix, show similarities.

2.2. Orientation-based App sniffing

The previous subsection has shown a possible approach to sniff Apps on a mobile phone based on magnetometer data. The accuracy is about 0.84. We will introduce enhanced schemes which achieve a higher accuracy of close to 1 in the next section. But before that, we would like to discuss how an attacker can obtain sensory data from a user's smartphone.

By default, any App on a smartphone can access magnetometer without user permission. So the most straightforward approach is to embed a small piece of code in an benign App to report magnetometer data to the attacker. However, not all mobile users would install such App. Toward this end, we have further considered a web-based method, where the attackers can acquire sensor data by simply integrating a small (4 line) Javascript code in their webpage. When a smartphone browses the webpage, the Javascript can read a range of sensory data such as gyroscope and accelerometer that we will use later. However, it cannot attain direct access to the magnetometer of the mobile devices.

```
function deviceOrientationHandler(eventData) {
  var ori_gamma = eventData.gamma;
  var ori_beta  = eventData.beta;
  var ori_alpha = eventData.alpha;
}
```

Nevertheless, our investigation reveals that the orientation of the devices can be sniffed using the web-based method. Furthermore, the orientation is closely correlated with the magnetometer data as shown by comparing Figs. 5 and 1(c). As a matter of fact, the primary use of magnetometer data on the smartphone is to calculate the device's orientation in conjunction with the accelerometer. Based on this interesting observation, we use the orientation of the device as an

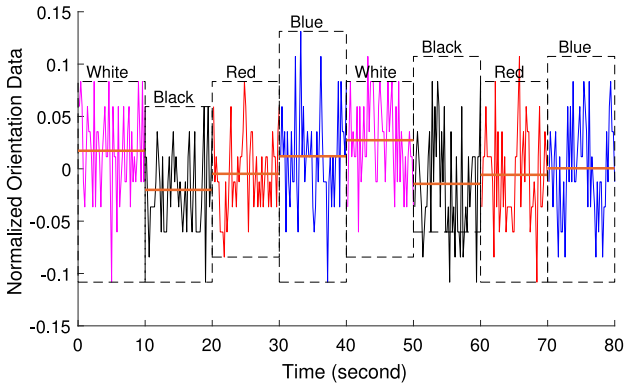


Fig. 5. Change of orientation data while the smartphone sequentially displays four different colors, white–black–red–blue.

Table 3
Performance Comparison of Different CNN Models with Tempogram.

Machine Learning Model	Accuracy
3-layer CNN	47%
5-layer CNN	48%
3-layer CNN with spectrogram	51%
5-layer CNN with spectrogram	51%
3-layer CNN with tempogram	87%
5-layer CNN with tempogram	87%
6-layer CNN with tempogram	**90%**
7-layer CNN with tempogram	89%
8-layer CNN with tempogram	90%

Table 4
Performances of clustering based on motion sensors.

No. of grids	4	8	12	16	20	24	28
Accuracy	100%	100%	99%	99%	98%	98%	98%

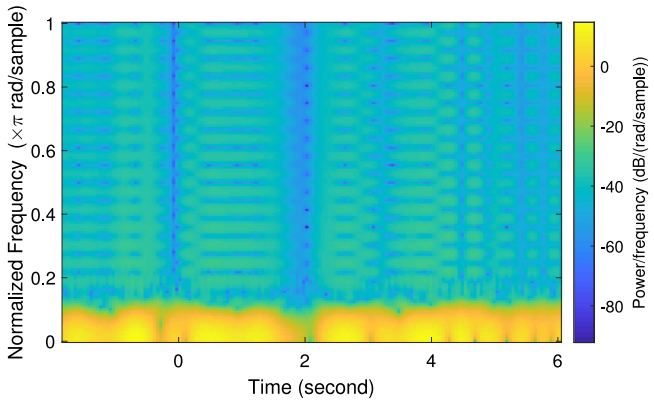
alternative to train and test the CNN model. The performance (i.e., recognition accuracy) is only slightly lower than the original approach based on magnetometer data. Details results will be discussed in Section 4 (see Table 5 and related discussions).

2.3. MAS in time–frequency representation

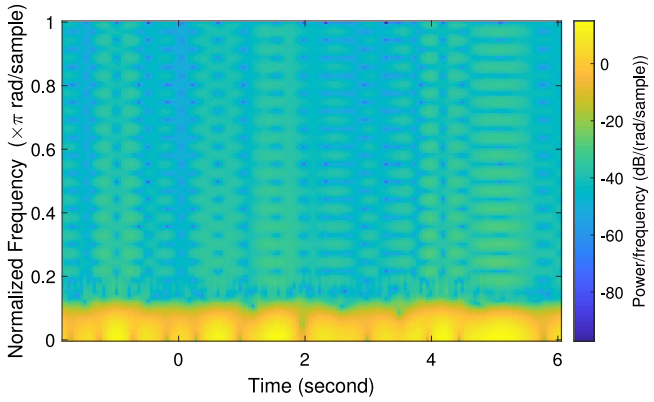
In the above discussion, we have demonstrated a side channel attack, i.e., MAS, where the magnetometer data on a smartphone are exploited by the attacker to infer the Apps opened by the mobile user. The attack is implemented by using magnetic signals in time domain, achieving an accuracy of about 84%. Next, we further demonstrate the exploitation of frequency domain signals that can potentially launch a more effective attack.

The frequency domain has been widely utilized in modern signal processing. In our case, since the magnetic signal varies along the time domain, we need to consider the time and frequency domain simultaneously, naturally leading to the use of spectrogram – a graph of the spectrum of frequencies of a signal as they vary with time. The spectrogram is an accurate representation of audio signal since human hearing is based on real-time spectrogram encoded by ear. It has been extensively utilized in audio signal processing [23,24]. Fig. 6 shows two examples of the spectrogram graph of the magnetic sensor readings while opening HBO and Google Maps, respectively.

A naive approach is to directly replace the time domain signals by spectrogram, and then train CNN models based on the input of spectrogram of magnetic sensor readings. The testing results are shows in Table 3. The performance of the CNN models with spectrogram input is slightly higher than the CNN models with raw magnetic time domain signals (see Table 1), but significant lower than the MAS with slicing, which is around 84%. The poor performance is not unexpected, because the frequency of the magnetic sensor reading majorly ranges from 2 to 10 Hz, leaving the majority area of the spectrogram graph to be blank. Therefore, the differences between spectrogram graphs are largely concentrated in a very small area which leads to confusion and misclassification of the CNN model. We have conducted a similar experiment by limiting the maximum frequency to 10 Hz, in particular, to filter out the un-needed frequencies. However, the subtle change of the magnetic sensor cause by LED display still cannot be efficiently captured by the spectrogram which leads to non-differentiable images. To this end, we have explored an alternative approach based on Tempogram.



(a) Spectrogram of HBO.



(b) Spectrogram of Google Maps.

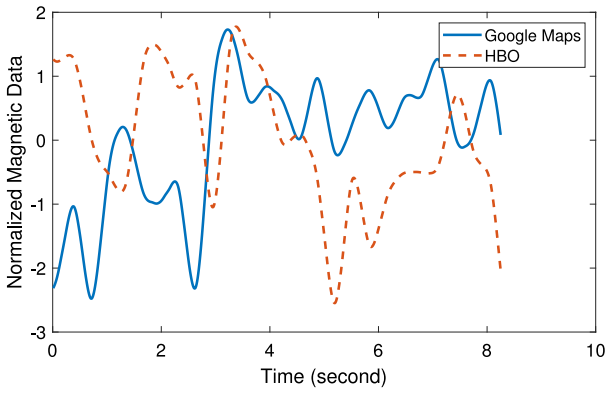
Fig. 6. Examples of spectrogram.

Table 5
Performance Comparison of MAS Approaches.

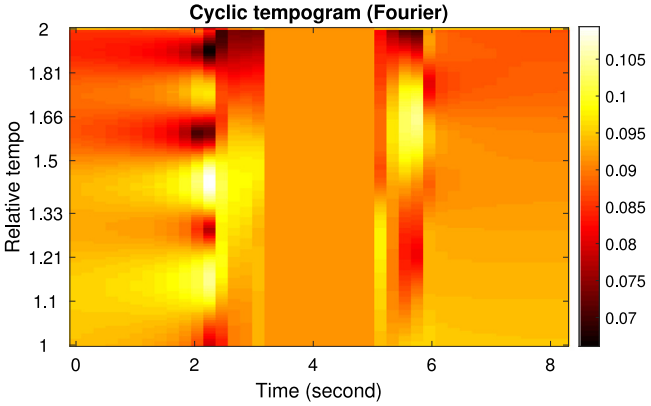
Number of Apps	3	7	11	15
Magnetic Model (Single Device)	93%	90%	86%	84%
Magnetic Model (Cross Device)	91%	87%	84%	82%
Magnetic Model (Cross Model)	72%	68%	62%	59%
Magnetic Model (Cross Model Mix)	91%	87%	84%	81%
Magnetic Model (Cross Model Mix) + Downsampling	91%	87%	84%	81%
Magnetic Model (Cross Model Mix) + Upsampling	91%	88%	83%	81%
Orientation Model (Cross Model Mix)	84%	80%	79%	74%
Magnetic Model (Cross Model Mix) + Motion	99%	98%	98%	97%
Orientation Model (Cross Model Mix) + Motion	98%	98%	98%	98%

Tempogram [25] is initially developed as a state distribution graph for music signals. It was originally designed to extract local tempo and beat information from audio recordings. In our case, since the subtle change of the magnetic signal caused by the LED screen is similar to the beats in audio signal, we expect the tempogram graph of the original magnetic signal can better extract the features of the magnetic signal's subtle changes. Fig. 7 shows the raw magnetic sensor data and its corresponding tempogram graph. Specifically, we convert a 1-D sensor reading to a 2-D graph with patterns which is easier for CNN to recognize. Since CNN has proven success in image recognition, we expect this will lead to improved performance.

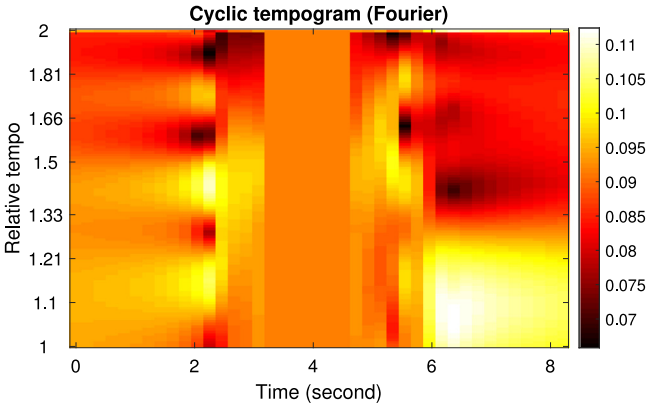
Similar to the earlier discussions, a deep CNN is devised and trained by using the tempogram graphs as inputs. We have conducted a series of experiments. As shown in Table 3, the recognition accuracy of CNN using tempogram is improved to 90% which is dramatically higher than the raw magnetic data and spectrogram. It is also higher than the slicing approach which is 84% (see Table 1). In the meantime, it decreases the size of training dataset by 90% in comparison with the slicing approach, and accordingly reduce the total computation time significantly.



(a) Magnetic data.



(b) Tempogram of HBO.



(c) Tempogram of Google Maps.

Fig. 7. Magnetic reading and tempogram.

3. Motion sensor-assisted MAS with tempogram

In the previous section, we have demonstrated a side channel attack, i.e., MAS, where the attacker sniffs the magnetometer data on a smartphone and accordingly infers the Apps opened by the mobile user. The accuracy of such inference can be 90%. In this section, we show that the attack can be worse, i.e., the attacker can achieve even higher accuracy, if he exploits motion sensor data.

Briefly, on a given smartphone, the locations of the Apps are generally fixed during the time window when the attacker sniff sensor data (e.g., ranging from a few hours to a few days). Suppose the mobile user clicked on Chase App 20 times

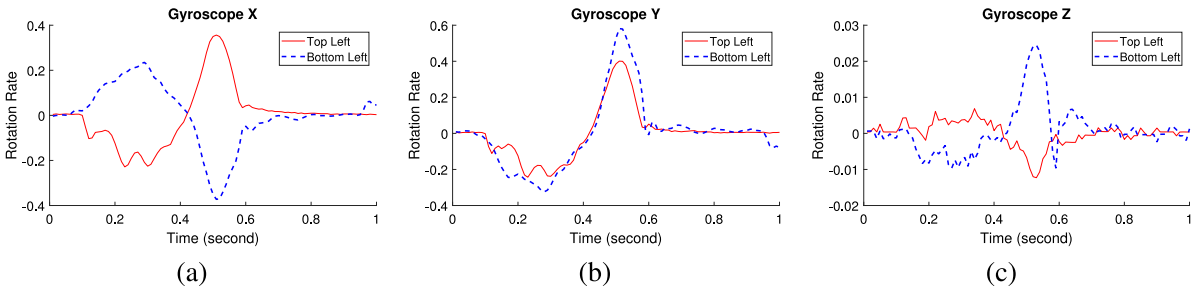


Fig. 8. Comparison of gyroscope data on X, Y, and Z axis while clicking on top-left and bottom-left of the screen.

during the period. If the 20 magnetometer or orientation data are fed into the CNN model introduced in Section 2, about four of them would be classified incorrectly to some other Apps. However, if the attacker is able to recognize that the 20 clicks are all at the same location on the screen, then he can put them into a cluster and feed the cluster of magnetometer data to the CNN model. The vast majority of them (90% in average) should be recognized correctly as Chase. Since the cluster of clicks are from the same location, they should be the same App.[1] Therefore, the attacker can conclude that all 20 clicks are Chase. This approach is effective because, as to be shown next, such clustering can be achieved with high accuracy (nearly 100%) by using motion sensor data on the phone.

3.1. Clustering based on motion sensor data

Nowadays, most mobile operating systems place their App icons in a fixed grid layout. For instance, iPhone 7 plus has a 4×7 layout and Samsung S7 uses a 4×5 grid. When a user clicks on different spots on the touch screen, the smartphone has a small rotation and/or vibration that can be captured by the 3-axis gyroscope and accelerometer. Similar to the discussion on magnetometer, different phones sample their motion sensors at different frequencies: 100 Hz on iPhone and 50 Hz on Samsung Galaxy. Fig. 8 illustrates the 3-axis gyroscope data while clicking on the top-left and bottom-left on an iPhone's screen. As can be seen, the gyroscope waveforms differ on all three axes, especially the X and Z axes. Several previous studies have shown the use of motion sensors to infer the touches on a smartphone screen [26].

The attacker can build a training data set for each popular smartphone model and feed the training data to CNN to create a motion classifier for this type of smartphone. Note that, we cannot mix the training data of iPhone and Samsung phone since their layout are different. Again, various CNN architectures and hyperparameters are explored, and finally we adopt a 4-layer CNN model with the following setting in our experiments: *Conv*(64)-*Conv*(64)-*Pool*-*Dense*(128)-*Dropput*(0.5)-*Sfmax*.

As discussed earlier, motion sensor (including accelerometer and gyroscope) data are freely accessible by Apps or web-embedded Javascript. For training purpose, the attacker can easily experiment on different phones to build training data sets. For example, given a type of phone, the attacker can click on every grid points for a number of times, collect motion sensor data, and label each data with the corresponding grid point. The 3-axis gyroscope data and accelerometer data are combined together. A click lasts about 1 s. So, each data has a dimension of 6×100 on iPhone since its sampling rate is 100 Hz, or 6×50 on Samsung S7 that samples at 50 per second.

The CNN models are trained offline. Once they are ready, the attacker can acquire motion sensor data from mobile users either via Apps installed on the users' phones or when the mobile users browse the attacker's websites embedded with his Javascript as we discussed earlier. The data from each user are fed into the selected CNN model that matches the user's phone. If the type of phone is unknown, the attacker can always try different CNN models and choose the one that yields the most reasonable result. Thus, the attacker can label each click with a grid position on the screen, and accordingly group all clicks with the same label into a cluster.

For example, in our preliminary experiment, we have collected gyroscope and accelerometer data when clicking 100 times on each spot of an iPhone 7 Plus's screen. The experiments have been repeated by two people using both right and left hands. In total, dataset contains $2 \times 2 \times 100 \times 4 \times 7 = 11200$ data samples and each data sample has a dimension of 6×100. Again, 4-fold cross validation is adopted.

Table 4 shows the clustering results. When we only consider four possible positions (at the four corners of the screen), the accuracy is perfectly 1.0. With the increase of grid points, the accuracy decreases slightly but is still maintained stable around 98%. With such high accuracy, the attacker can effectively group the clicks into clusters and consider all clicks in the same cluster to be associated with the same App.

[1] We have assumed that the most frequently used Apps are on the home page. The scenarios with multiple pages and groups will be investigated in our future research.

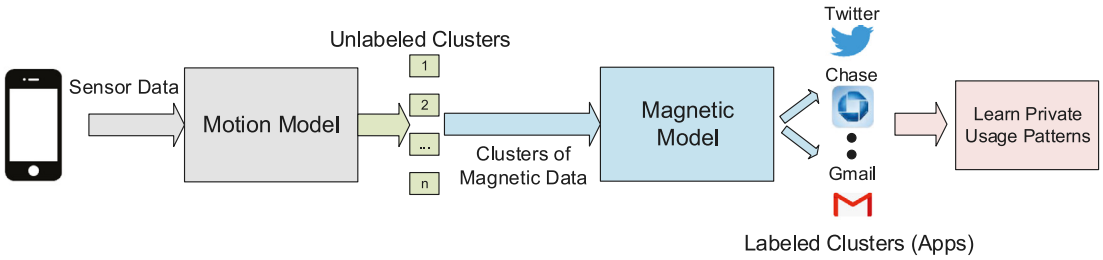


Fig. 9. The overall procedure of the motion sensor-assisted MAS.

3.2. Motion sensor-enhanced attack

Based on the above findings, we now combine the clustering scheme (enabled by motion sensors) and the App classification based on magnetometer or orientation data. The overall procedure of the motion sensor-assisted MAS is illustrated in Fig. 9.

Assume that a smartphone has visited the attacker's website embedded with the aforementioned Javascript. Since the Javascript has free access to accelerometer, gyroscope and orientation, the attacker can continuously eavesdrop such sensor data. Note that, even the webpage is in the background, the Javascript can still acquire data. Of course, the attacker can also try to camouflage codes in seemingly benign Apps, given the behavior of accessing the sensor data is fairly common.

The hacker continuously collects sensor data for a desired period (usually for several hours or days), and then performs a straightforward preprocessing to identify the clicks on the screen and format corresponding motion data and tempogram of magnetic (or orientation) data as follows:

$$\begin{pmatrix} click_1 & motion_1 & tempogram\_of\_mag_1 & time\_stamp_1 \\ click_2 & motion_2 & tempogram\_of\_mag_2 & time\_stamp_2 \\ \vdots & \vdots & \vdots & \vdots \\ click_n & motion_n & tempogram\_of\_mag_n & time\_stamp_n \end{pmatrix}$$

Each row represents one data comprising a click ID, corresponding motion sensor recording, magnetic or orientation recording, and other information such as the time stamp. The attacker first uses the 4-layer motion CNN model to classify each click (i.e., each row of the dataset) into a cluster and label it with the corresponding grid ID. The maximum number of clusters equals to the number of grids on the mobile screen. Note that, all clicks in the same cluster are from the same grid location and thus should be the same App. As shown in Table 4, this step is very accurate.

Next, the attacker feeds a cluster of tempogram of magnetic (or orientation) data with the same grid ID to the 6-layer tempogram-magnetic CNN model. Over 80% of them are expected to be classified correctly, according to the accuracy presented in Table 1. Because they all belong to the same App, a majority vote can effectively determine the App for the entire cluster.

As a result, the attacker can infer what Apps have been installed on the user's mobile device, how frequently they are used, and when they are opened. In other words, the attacker can track the users' habits of App usage. The attack can become even worse. For example, if the attacker has identified a bank App, it is not too hard to capture the motion sensors while the user types username and password. If the keyboard is static, the attacker can obtain the password by using a similar approach as discussed above and access the bank account.

4. Experimental evaluation

We have used iPhone 7 Plus and Samsung Galaxy 7 to implement and demonstrate the attacks. In this section, we present our experiments and results.

4.1. System setup

To collect motion sensor training data, we have considered iPhone 7 Plus and Samsung S7 separately, since they have different App layouts: 4×7 and 4×5. For each grid point in a layout, we have collected 400 samples. The corresponding dataset size for each model is $400 \times 4 \times 7$ and $400 \times 4 \times 5$.

To recognize Apps, we have considered both magnetometer and orientation data. As discussed before, the latter is desired because it can be accessed by Javascript embedded in the attacker's website. To create the training dataset, we have considered top-15 most used Apps. We open each App 100 times and record the corresponding magnetometer and orientation data.

Table 6
Accuracy under magnetic interferences..

Distance to refrigerator (cm)	25	50	100
Magnetic Model (Cross Model Mix) + Motion	97%	97%	98%
Orientation Model (Cross Model Mix) + Motion	98%	97%	98%

All CNN models are implemented in *Tensorflow* [19]. The training is done offline and the training data can be easily obtained by the attacker himself. On the other hand, it is trickier to collect sensor data of the victims. To this end, we have developed a mobile webpage embedded with a sensor data collection Javascript (for gyroscope, accelerometer, and orientation data) and a sensor data collection App on both iOS and Android (for gyroscope, accelerometer, and magnetometer data). We run the experiment for one day to collect data from users. Some clicks are to open Apps while others happen in the Apps. We differentiate them by detecting the pressing of the home button on iPhone or Samsung phones. We only process the sensor data of the clicks after pressing home button. Note that, the attacker is not necessary to process all clicks. As long as he can collect enough number of useful sensor data, he can achieve his goal of sniffing users' Apps and tracking their usage.

We have shown some initial experimental results in Section 2, assuming only magnetometer data are sniffed to infer Apps. More results are presented here by considering different number of Apps and different MAS approaches. In the following discussion, "Magnetic Model" denotes the baseline method where only the tempogram of magnetometer data are used for training and testing; likewise, "Orientation Model" means the tempogram of orientation data are used for training and testing; "+ Motion" indicates motion sensor data are also employed by following the procedure presented in Fig. 9.

We have also experimented on various devices. "Single Device" denotes the experimental setting where training and testing are carried out based on the data from a single device; "Cross Device" indicates the experiments conducted in a way that training is based on the data from a device, while testing is on a different device but of the same type (e.g., both devices are iPhone 7 Plus or both are Samsung Galaxy 7); "Cross Model" shows the results where training is based on the data from a device, but testing is on a different device of the different type; finally, "Cross Model Mix" means the setting where we mix data from different devices in different models for both training and testing. Note that, the sensors' sampling rates on iPhone 7 Plus and Samsung Galaxy 7 are different. So, under "Cross Model Mix", the training and testing datasets include data sampled at different rates. "+Downsampling" and "+Upsampling" are the signal processing schemes to decrease the sampling rate on iPhone 7 Plus or increase the sampling rate of Samsung Galaxy 7 to keep them consistent.

4.2. Experimental results

As can be seen in Table 5, the accuracy generally decreases when more Apps are considered. When there are more Apps, their feature distances become shorter, thus resulting in higher errors in CNN classification. For a given number of Apps, the accuracy is the lowest under "Magnetic Model (Cross Model)". This is because different smartphone models (especially different manufacturers) often use different types of magnetometers. Therefore, if we train the CNNs based on data from iPhone 7 Plus, but test them on Samsung Galaxy 7, the performance is naturally low. However, if training and testing are both based on mixed data from different device models, the performance degradation becomes negligible (see "Magnetic Model (Cross Model Mix)"). It is straightforward for an attacker to collect data from various popular phones for training purpose. In addition, "Upsampling" and "Downsampling" do not significantly improve the performance. The CNN model is not sensitive to the variance of sampling rate.

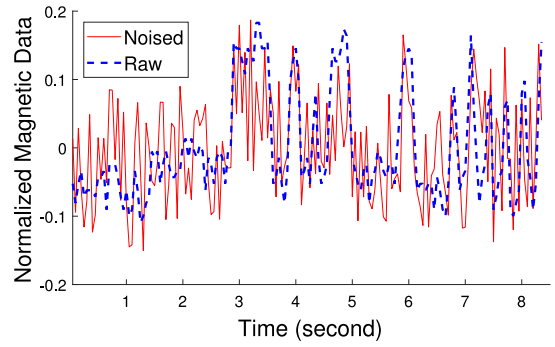
Comparing "Magnetic Model (Cross Model Mix)" and "Orientation Model (Cross Model Mix)", the latter's accuracy is only lower than the former by 4–6%. As discussed in Section 2, orientation is highly correlated with magnetometer data. The use of orientation makes the attack much easier, given that it can be obtained by a Javascript embedded in the attacker's webpage.

When motion sensor-assisted MAS is employed, the accuracy becomes as high as 98%, and the difference between magnetometer and orientation data are fading away. This is because the difference in their accuracy does not affect the majority vote.

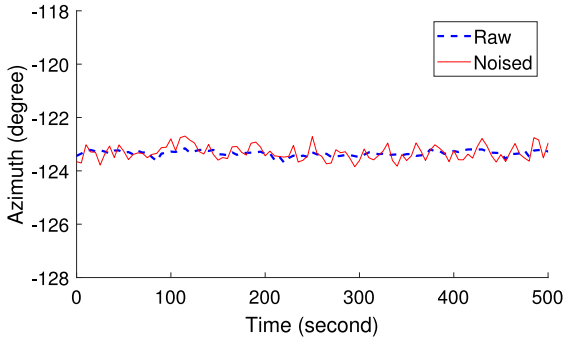
In addition, large electronic devices, for example, refrigerators, may generate interference on the magnetic field. To this end, we have carried out experiments when the smartphone is placed at different distances to a refrigerator. As shown in Table 6, the impact is insignificant. This is because the interference is a constant and thus canceled out after normalization.

5. Defense mechanism

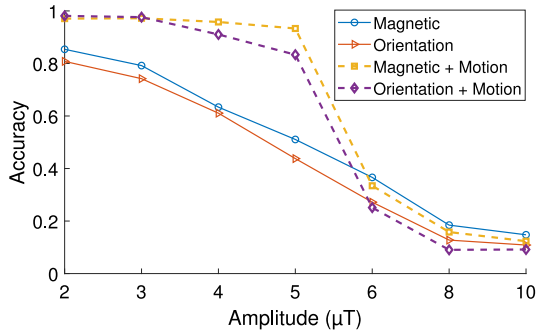
In this section, we discuss viable methods to mitigate the MAS attack. The first approach is to restrict the permission to access magnetic, orientation and motion sensors. With this method, a system notification will be popped up when an



(a)



(b)



(c)

Fig. 10. (a) Noised vs. original magnetic data. (b) E-compass data based on noised and original magnetometer, where the noise is at the level of 8 μ T. The resulting error of e-compass reading is only 0.2° in average. (c) MAS accuracy under different noise amplitude.

APP or Javascript requests access to those sensor data, alerting the users. Unfortunately, despite a potential threat, users may still obliviously permit such access.

A more transparent method is noise injection that perturbs magnetic sensor output. Since the change of magnetic field caused by LED is relatively small, a minor Gaussian noises can be introduced into the magnetometer or orientation data to mitigate the attack. An example of such noise is illustrated in Fig. 10(a). It has minimum impact on normal applications. For example, the e-compass uses magnetometer data to determine how many degrees the phone's front deviates from the true North. Its results based on noised and original magnetometer signals are shown in Fig. 10(b), with the mean error of only 0.2°. For most applications, such small errors do not substantially affect their functionality.

On the other hand, the noise will significantly affect the accuracy of MAS. For instance, we generate a Gaussian noises at different levels (from $2 - 10$ μT) and observe their impact on MAS accuracy.

As shown in Fig. 10(c), the accuracy of the magnetic model drops to 15% when the average amplitude of noise is 10 μT. Similarly the accuracy of orientation model degrades significantly with the increase of noise level. Under the motion sensors-assisted MAS which exploits motion sensor data for clustering, it is even more interesting to observe the sharp accuracy decrease when the noise level exceeds 5 μT. In this case, the majority vote is likely wrong, thus the entire

cluster is classified incorrectly. One possible attack against our defense mechanism is re-training in the presence of the noise. However, once the injected noise is around the same level of the magnetic signal, re-training cannot learn useful information from the output.

6. Discussion and future work

We have observed the subtle correlation between an LED display and its surrounding magnetic field. We have further demonstrated a new side-channel attack to smartphones by exploiting this correlation, where the attacker can sniff mobile Apps by analyzing magnetometer or orientation data along with motion sensor data using deep learning techniques. We have conducted extensive experiments on both iPhone 7 Plus and Samsung Galaxy 7 under different scenarios. Our experiments have demonstrated that the App sniffing accuracy is as high as 98%. At last, we have proposed a noise injection scheme to effectively mitigate such attacks.

6.1. Discussion

Our approach achieves a 98% success rate when the users are stationary and can be widely deployed to sniff millions of user by setting up and hijacking multiple websites. While users are moving, the readings of magnetometer and motion sensors can vary drastically. As the change of the magnetic field caused by LED is relatively small, the success rate of MAS may be low under human movements. However, MAS is able to continuously sniff users once it is set up. For frequently used Apps, we expect there are plenty of opportunities that users would launch Apps both at moving and being stationary. As long as the users launch the Apps while being stationary, MAS can accurately sniff the Apps. This is particularly effective if an attacker wants to steal credentials such as online banking accounts. Certainly, improving MAS's performance under human movements is an interesting problem for future direction. Several studies [16,17] showed that sensor readings are closely correlated with different human activities. For instance, stepping causes a repeatable pattern on motion sensors and magnetometer readings. Hence, the attacker may detect human activity at first and then tries to cancel out the drastic change caused by the human activity. This will be one of our future directions.

Our approach delivers a relatively lower recognition rate on Apps with similar colors and patterns. To address this problem, one possible solution is to extend the detection time window to obtain more unique features from those Apps. Furthermore, since users tend to behave uniquely (type, swipe, and pinch) in different Apps, the attacker may also leverage motion sensor readings to enhance the recognition rate.

It is also worth to mention that the nearby large electronic devices may interfere the magnetometer reading. When users are stationary to the nearby devices, the interference remains constant, which may be canceled out by the normalization of data preprocessing. We will explore a better solution when users are moving around large electronic devices in our future study.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the Office of Naval Research, USA and the National Science Foundation, USA under Grants CNS-1528004, CNS-1659795, CNS-1745632, CNS-1828593, EEC-1840458, OAC-1829771, and CNS-1528004.

References

[1] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, W. Xu, My smartphone knows what you print: Exploring smartphone-based side-channel attacks against 3d printers, in: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2016, pp. 895–907.

[2] A. Hojjati, A. Adhikari, K. Struckmann, E. Chou, T.N.T. Nguyen, K. Madan, M.S. Winslett, C.A. Gunter, W.P. King, Leave your phone at the door: Side channels that reveal factory floor secrets, in: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, CCS, 2016, pp. 883–894.

[3] M. Mehrnezhad, E. Toreini, S.F. Shahandashti, F. Hao, TouchSignatures: Identification of user touch actions and PINs based on mobile sensor data via JavaScript, J. Inf. Secur. Appl. 26 (2016) 23–38.

[4] H. Wang, T.T.-T. Lai, R. Roy Choudhury, Mole: Motion leaks through smartwatch sensors, in: Proceedings of the 21st ACM Annual International Conference on Mobile Computing and Networking, MOBICOM, 2015, pp. 155–166.

[5] Y. Chen, X. Jin, J. Sun, R. Zhang, Y. Zhang, Powerful: Mobile app fingerprinting via power analysis, in: Proceedings of IEEE International Conference on Computer Communications, INFOCOM, 2017, pp. 1–9.

[6] X. Zhao, M.Z.A. Bhuiyan, L. Qi, H. Nie, W. Rafique, W. Dou, TrCMP: An app usage inference method for mobile service enhancement, in: Proceedings of International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage, SpaCCS, 2018, pp. 229–239.

[7] Q. Wang, A. Yahyavi, B. Kemme, W. He, I know what you did on your smartphone: Inferring app usage over encrypted data traffic, in: Proceedings of IEEE Conference on Communications and Network Security, CNS, 2015, pp. 433–441.

[8] V.F. Taylor, R. Spolaor, M. Conti, I. Martinovic, Robust smartphone app identification via encrypted network traffic analysis, IEEE Trans. Inf. Forensics Secur. 13 (1) (2018) 63–78.

[9] S. Narain, T.D. Vo-Huu, K. Block, G. Noubir, Inferring user routes and locations using zero-permission mobile sensors, in: Proceedings of IEEE Symposium on Security and Privacy, SP, 2016, pp. 397–413.
[10] A. Das, N. Borisov, M. Caesar, Tracking mobile web users through motion sensors: attacks and defenses, in: Proceedings of the Network and Distributed System Security Symposium, NDSS, 2016, http://dx.doi.org/10.14722/ndss.2016.23390.
[11] LED color guide, http://www.lumex.com/article/led-color-guide.
[12] [Online] Available at:. https://www.mathworks.com/help/wavelet/ref/wden.html.
[13] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Proceedings of Advances in Neural Information Processing Systems, NIPS, 2012, pp. 1097–1105.
[14] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2015, pp. 1–9.
[16] M. Zeng, L.T. Nguyen, B. Yu, O.J. Mengshoel, J. Zhu, P. Wu, J. Zhang, Convolutional neural networks for human activity recognition using mobile sensors, in: Proceedings of International Conference on Mobile Computing, Applications and Services, MobiCASE, 2014, pp. 197–205.
[17] S. Ha, S. Choi, Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors, in: Proceedings of International Joint Conference on Neural Networks, IJCNN, 2016, pp. 381–388.
[18] Smartphone market share, http://www.prnewswire.com/news-releases/comscore-reports-april-2017-us-smartphone-subscriber-market-share-300471167.html.
[19] [Online] Available at:. https://www.tensorflow.org.
[20] C.-C. Chang, C.-J. Lin, LIBSVM – A library for support vector machines, Proc. ACM Trans. Intell. Syst. Technol. 2 (3) (2011) 27.
[21] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: Proceedings of European Conference on Computer Vision, ECCV, 2014, pp. 818–833.
[22] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information, 2017, arXiv preprint arXiv:1703.00810.
[23] Y. Masuyama, K. Yatabe, Y. Oikawa, Low-rankness of complex-valued spectrogram and its application to phase-aware audio processing, in: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2019.
[24] A.M. Badshah, J. Ahmad, N. Rahim, S.W. Baik, Speech emotion recognition from spectrograms with deep convolutional neural network, in: Proceedings of International Conference on Platform Technology and Service, PlatCon, 2017, pp. 1–5.
[25] M. Tian, G. Fazekas, D.A.A. Black, M. Sandler, On the use of the tempogram to describe audio content and its application to music structural segmentation, in: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2015, pp. 419–423.
[26] M. Mehrnezhad, E. Toreini, S.F. Shahandashti, F. Hao, Stealing PINs via mobile sensors: Actual risk versus user perception, Int. J. Inf. Secur. (2016) 1–23, http://dx.doi.org/10.1007/s10207-017-0369-x.