MAGTHIEF: Stealing Private App Usage Data on Mobile Devices via Built-in Magnetometer

Hao Pan†, Lanqing Yang†, Honglu Li†, Chuang-Wen You‡, Xiaoyu Ji§, Yi-Chao Chen†, Zhenxian Hu†, Guangtao Xue† *

† *Department of Computer Science and Engineering, Shanghai Jiao Tong University*
‡ *Interdisciplinary Program of Technology and Art, National Tsing Hua University*
§ *College of Electrical Engineering, Zhejiang University*
{panh09,yanglanqing,lhl980514,yichao,huzhenxian, gt_xue}@sjtu.edu.cn, cwyou@mx.nthu.edu.tw, xji@zju.edu.cn

***Abstract*—Various characteristics of mobile applications (apps) and associated in-app services have been used reveal potentially-sensitive user information; however, privacy concerns have prompted third-party apps to rigorously restrict access to data related to mobile app usage. This paper outlines a novel approach to the extraction of detailed app usage information based on analysis of the electromagnetic (EM) signals emitted from mobile devices when executing app-related tasks. Note that this type of EM leakage becomes high-complex when multiple apps are used simultaneously and is subject to interference from geomagnetic signals generated by device movement. This paper proposes a deep learning-based multi-label classification system to identify apps and in-app services based on magnetometer readings. The proposed MAGTHIEF system uses accelerometer and gyroscope data to cancel out the offset in geomagnetic signals followed by an elaborate deep region convolution neural network (DRCNN) to differentiate among multiple apps and the corresponding in-app services. Experiments on 50 apps demonstrated the efficacy of MAGTHIEF in identifying multiple apps and in-app services, achieving high average macro F1 scores of 0.87 and 0.95, respectively. MAGTHIEF also achieved time duration accuracy of 89.5% in recognizing app trajectory in the real-world scene.**

***Index Terms*—electromagnetic signal, side-channel sensing, mobile application usage**

I. INTRODUCTION

Background: Mobile devices have become an increasingly ubiquitous part in the daily life. Mobile app usage behavior varies with the personal interests and preferences of the user, even the in-app service usage patterns are diverse when using the same app. As shown in Fig. 1, app usage behaviors can uniquely reveal the detailed user privacy profiles. Many third-party app developers/service providers rely on this information for further applications, such as personalized services/advertisement recommendation [1], screen addiction monitor [2] and human mobility prediction [3]. However, concerns pertaining to the leakage of private information has prompted the third-part apps to curtail access to mobile app usage data via system-kernel information (*e.g.*, /proc filesystem, battery, and internet traffic data) on Android and iOS devices [4], [5]. Data from cellular network providers is another potential avenue by which to obtain app usage information; however, security protocols for in-app services are making this increasingly difficult. Governments have also begun establishing privacy-related regulations limiting third-party access to data with user labels.

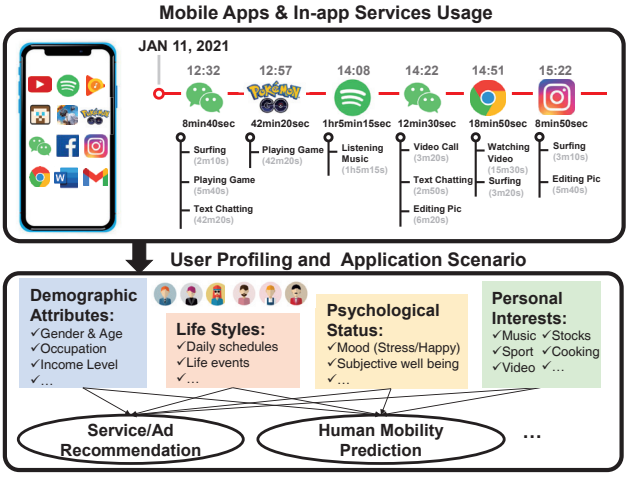


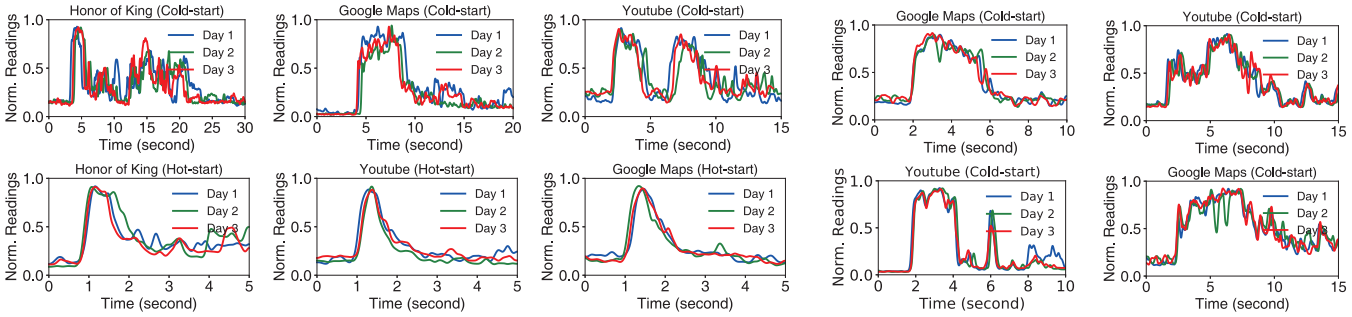
Fig. 1. Extraction of potentially-sensitive user information through analysis of detailed mobile app usage.

Existing EM-based methods and limitations: A number of researchers have explored the possibility of using electromagnetic (EM) signals to infer the launch of specific apps [6]–[9]. More specifically, these schemes exploited magnetometer readings to CPU activity to fingerprint apps being launched from scratch (via a "cold start") on the laptops and smartphones. In real-world scenarios involving mobile devices, the launching of apps varies. Apps launched in a "hot-start" manner (reloading data already residing in memory) are difficult to identify with the same method in [7] (see Sec. II-A), and systems designed to detect the launching of mobile apps are unable to track the detailed time durations of active apps with which the user is closely interacting in the foreground or identify apps running in the background. Furthermore, much of the information required to infer personal behavior and preferences pertains to in-app services usage [10].

Proposed Scheme: This paper proposes a novel system based on the EM side channel for the characterization of mobile app usage behavior in real time. The proposed MAGTHIEF system continuously collects the built-in magnetometer

* Corresponding author.

978-1-6654-4108-7/21/$31.00



(a) Using Oppo Reno4 Pro (b) Using Huawei P20Pro (c) Using iPhone 7Plus

Fig. 2. EM patterns obtained from three smartphones while launching different apps in cold- or hot-start manners.

readings with the aim of inferring when the user is accessing a specific app based on the corresponding in-app service, as well as identifying other apps/in-app services running in the background simultaneously.

Challenges: Development of the MAGTHIEF system imposed a number of daunting challenges: **(1)** The built-in magnetometer captures EM signals associated with running apps from the target device as well as fluctuations in the surrounding geomagnetic field. Thus, our first challenge was to extract EM signals from noisy magnetometer readings. **(2)** Mobile devices are capable of handling with multiple app tasks in both the foreground and background, which bring the second challenge to differentiate each running app among the high-complex EM signals. **(3)** Many apps provide various in-app services, which makes it difficult to identify the specific app running at any given time. Developing an explicit model by which to identify both the in-app service label as well as the corresponding app label is the third challenge. **(4)** There are at present a staggering number of mobile devices on the market, and the unique configuration of each may generate different EM patterns, even when performing the same app tasks. Thus, our final challenge was to characterize similarities and differences in the EM patterns emitted from various mobile devices.

Solutions: The aforementioned challenges were addressed as follows: **(1)** We first developed a multi-layer perceptron (MLP) regression scheme, which uses 3-axis accelerometer and 3-axis gyroscope data to identify geomagnetic signals associated with device movement. **(2)** We developed a deep learning model to handle the classification of multiple app-related tasks. We first designed shared convolution layers for the generation of feature maps, and a region proposal network (RPN) for the detection of candidate apps. Each candidate app region is treated as a region of interest (ROI), and the corresponding ROI feature maps are resized to fit the app classification structure. **(3)** We also designed a cascaded classification model to characterize in-app service and app type. The ROI feature map of each app is first fed into average pooling and fully-connected layers for the generation of in-app service labels. The ROI feature map is also transformed via a 1×1 convolution layer, then fed into subsequent average pooling and fully-connected layers for app classification. **(4)** Preliminary experiments revealed that most mobile devices using a given operating system (OS) generate similar EM signals when performing the same tasks. In other words, the EM signals are affected primarily by the software environment rather than the internal hardware components. Thus, we trained two classification models respectively for each of the mainstream mobile OSs (Android and iOS).

The main contributions of this work are as follows:

- The proposed MAGTHIEF system is able to infer fine-grained mobile app usage information continuously via readings from the built-in magnetometer, without the need for user permissions.
- We developed a Deep Region Convolutional Neural Network (DRCNN) to facilitate the multi-label classification of multiple running apps running, and identify the corresponding in-app services.
- Extensive experiments demonstrated the efficacy of the proposed MAGTHIEF system, and it achieves high average macro F1 scores of $0.87/0.95$ when identifying multiple apps/in-app services respectively. MAGTHIEF also achieves up to 89.5% time duration accuracy in recognizing app trajectory in the real-world scene.

II. PRELIMINARY ANALYSIS

Preliminary experiments were conducted to answer three primary questions: i) Does the manner in which an app is launched affect the corresponding EM patterns? If so, can all of the EM patterns be distinguished? ii) What are the characteristics of the EM signals generated when using specific apps involving different in-app services? iii) What other factors impact magnetometer readings? Our answers to the above questions demonstrate the feasibility of using EM data to infer details pertaining to the detailed app usage.

Proprietary apps were installed on the devices to enable the continuous recording the built-in magnetometer at maximum sampling rates. The EM signals emitted from the device were calculated using the total magnetic field intensity from the three-axis magnetometer readings, as follows:$mag_t(t) = \sqrt{mag_x(t)^2 + mag_y(t)^2 + mag_z(t)^2}$ [7].

A. EM signals associated with launching an app

We first recorded EM signals generated when 10 apps were launched in cold-start (*i.e.*, from the hard disk) and hot-start

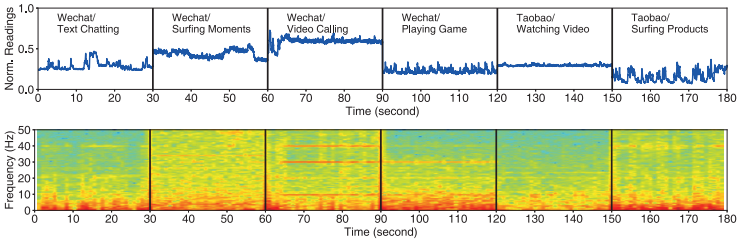


Fig. 3. Magnetometer readings and corresponding spectrograms generated while using various apps and in-app services.

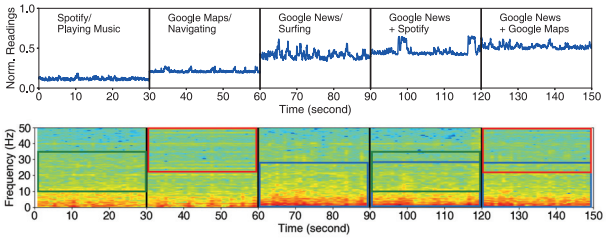


Fig. 4. Magnetometer readings and corresponding spectrograms generated while using multiple apps simultaneously.

TABLE I
CLASSIFICATION RESULTS OF EM SIGNALS GENERATED DURING COLD-/HOT-START LAUNCH OF TEN APPS.

	kNN	LDA	SVM	RF	MLP
Cold	89.7%	93.5%	93.7%	94.9%	95.6%
Hot	11.67%	12.92%	13.37%	15.72%	16.14%

(*i.e.*, from the "Recent Apps" list) manners. As shown in Fig. 2(a), the EM patterns presented good spatial and temporal consistency, varying only in the means by which the app was launched. This can be attributed to the fact that cold starts require comprehensive app initialization operations, whereas hot starts involve the reloading of data already residing in memory.

We applied the feature extraction method proposed in [7] to 20 EM signal traces obtained from each of the 10 apps launched in cold-start and hot-start manners. Five classic time-series classification algorithms: k-nearest neighbors (kNN), linear discriminant analysis (LDA), support vector machine (SVM), random forest (RF), and multilayer perceptron (MLP), were used to assess the EM patterns in terms of distinctiveness. The results of which are listed in Table I. These initial results classifying EM signal traces corresponding to hot starts were unsatisfactory, barely exceeding the accuracy of random guesses. In conclusion, the EM side channel provides little information pertaining to the hot-start launch of apps.

B. EM signals across devices

The cold-start launch of an app involves the execution of a fixed code set, which can be used to facilitate the exploration of similarities and differences in EM patterns emitted from different mobile devices. Thus, we recorded the magnetometer readings generated by another two smartphones while the same version of the same app was launched, the results of which are shown in the cold-start part of Fig. 2(a) and Figs. 2(b)-2(c). We found that devices with the different OSes produced totally different EM patterns when performing the same app task, while devices with the same OS produced relatively similar EM patterns. This prompted us to train two classification models for each of the mainstream mobile OSs (Android and iOS). Note that the two models should share the same architecture, but the parameters are different.

C. EM signals involving in-app services

EM signals also vary when use uses different in-app services in the same app. Fig. 3 presents EM patterns and the corresponding spectrograms generated while using WeChat (a social/communication app) and Taobao (a shopping app). The resulting EM patterns showed that the functional differences between the apps result in different EM patterns. We can also see in Fig. 4 that the internal EM patterns can be distinguished according to the in-app services. Taken together, these results indicate that the EM side channel leaked information pertaining to app usage behavior as well as in-app services.

D. EM signals associate with execution of multiple apps

Multi-window schemes (*i.e.*, split-screens) make it possible to run multiple mobile apps simultaneously. For example, a user could be playing a game in one window, while browsing the web in another window. Note however that the multi-window mechanism does not alter the app activity lifecycle. Only the app with which the user has most recently interacted is active (*i.e.*, RESUMED state) at any given time. This means that the apps with which the user is not currently interacting do not generate EM interference signals.

Apps that run continuously in the background (*e.g.*, playing music/video, downloading files, or navigating) also generate EM signals, which could potentially interfere with the EM signals caused by active foreground apps. We recorded the EM signals when a user was surfing news in the foreground, while listening music or navigating in the background, and the corresponding EM patterns and spectrograms are shown in Fig. 4. In order to understand the characteristic of EM signals generated by multiple running in-app services, we adopted a convolutional neural network (CNN) with an attention module (*e.g.*, STN [11]) to determine areas on the spectrogram with the highest importance for the in-app service classification task. The green/red/blue box in the Fig. 4 represents the resulting important feature region of playing music/navigating/surfing respectively. To sum up, the EM patterns and spectrograms generated while multiple apps (especially the active in-app services) are running simultaneously are similar to the super-position of EM signals associated with each in-app service, particularly in the frequency domain. This should make it possible to break down complex EM signals into separate EM signals associated with specific apps/in-app services.

E. Other factors affecting magnetometer readings

Magnetometers were originally included in mobile devices to sense geomagnetic signals for the electronic compass. Thus, any change in the position and/or orientation of the

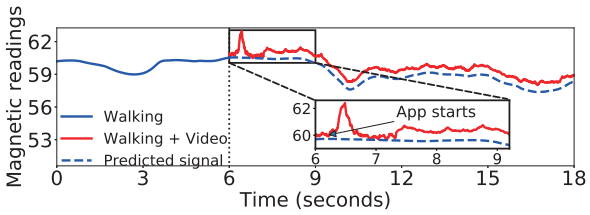


Fig. 5. Magnetometer readings obtained from mobile device while the user was walking with and without a video app playing.

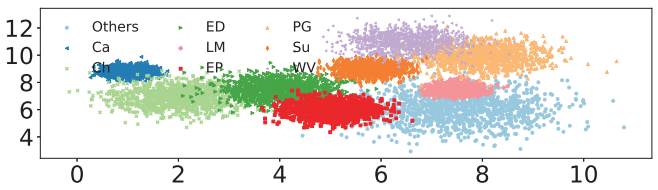


Fig. 6. Clusters related to nine types of in-app service. The notation is as follows: video/voice calls (Ca), text chatting (Ch), editing documents (ED), listening to music (LM), editing photos (EP), playing games (PG), surfing/reading (SU), watching video (WV), and Others.

device produces a corresponding change in the magnetometer readings. Fig. 5 illustrates the total magnetometer readings obtained while the user was walking. The blue line in Fig. 5 indicates changes in the geomagnetic signal caused by the movement of the user. The red line indicates the EM signals generated by the smartphone while performing app tasks. We discovered that changes in the amplitude of the geomagnetic signal induced by user movement exceeded the amplitude of EM signals emitted by the working device. This is a clear indication that changes in the geomagnetic signal can have a huge impact on data preprocessing (*i.e.*, normalization). The dashed blue line in Fig. 5 is the predicted geomagnetic field signals(see Sec. III-A), which can be used for geomagnetic fluctuation cancellation.

III. System Design

This section presents an overview of the proposed MagThief system comprising three key modules: data collection, preprocessing, and multi-label classification. Data collection involves the transfer of the built-in magnetometer and IMU sensor readings to a server for analysis. Preprocessing involves the use of 6-axis IMU sensor data to cancel out fluctuations in the geomagnetic field caused by device movement. Multi-label classification transforms EM signals into spectrograms and uses a deep learning algorithm to identify the app(s) running on the device and associated in-app services. Preprocessing and multi-label classification are detailed in the following.

A. Cancelling geomagnetic fluctuation

We first sought to resolve offset in the geomagnetic field signals caused by device movement imposed by the user walking or travelling in a vehicle. Ten volunteers were pre-assigned mobile devices to use in their daily lives for a testing period of one week, during which motion sensor data and magnetometer readings were recorded continuously. Note that no other apps were used during the data collection period. We then assessed six canonical regression algorithms: autoregressive integrated moving average (ARIMA), linear regression (LR), random forest regression (RFR), support vector regression (SVR) and multi-layer perception (MLP), in terms of geomagnetic signal fitting using 6-axis motion sensor data as the input and the total magnetic field strength as the output, the Mean Squared Error results (the smaller value means the better performance) of which are listed in Table II. Multi-layer perception (MLP) was the standout among these methods, due primarily to its ability to capture linear as well as nonlinear relationships (see the blue dashed line in Fig. 5). We therefore adopted MLP for the translation of 6-axis motion sensor data into geomagnetic field data (see the data preprocessing step in Fig. 7). Note that the MLP network architecture in this study combined an input layer with four hidden layers (respectively with 64, 64, 32, 16 neurons) and an output layer.

TABLE II
Results of geomagnetic signal fitting using canonical regression algorithms.

	ARIMA	LR	RFR	SVR	MLP
MSE	1.45	1.25	0.82	0.48	0.26

B. In-app service clustering and relabeling

The number of in-app service labels was impractical for classification; therefore, we performed time-series cluster analysis on the EM data using the TSkmeans algorithm [12] with the aim of formulating general categories pertaining to in-app functions. The massive quantity of historical data used in this study made it possible to aggregate the datapoints into nine categories, as shown in Fig. 6. We redefined a new label for each cluster, in accordance with the shared characteristics of each sample in the cluster. The clustering labels were then used to assign new labels to the original in-app service EM data to facilitate the training of the multi-task classification models for apps and in-app services.

C. Multiple apps/in-app services classification algorithm

As described in Sec. II-D, users commonly run multiple apps simultaneously. Thus, we designed a Deep Region Convolutional Neural Network (DRCNN) for the multi-label classification of multiple apps and in-app services, the architecture of which is shown in Fig. 7. The first step involves separating the EM features of each running app from the overall mixed EM signals using shared convolution layers to generate an EM feature map and a region proposal network [13] for the identification of candidate regions. The second step involves classifying each RoI candidate within a multi-label structure to output the in-app service and app labels.

1) Shared convolutional layers for feature generation: We determined that a mixture of EM signals from multiple apps could be represented as a superposition of individual apps in the form of a spectrogram. The frequency domain spectrograms used as classification model inputs in this study were calculated using the short-time fourier transform (STFT). The EM spectrograms were fed into shared convolution layers

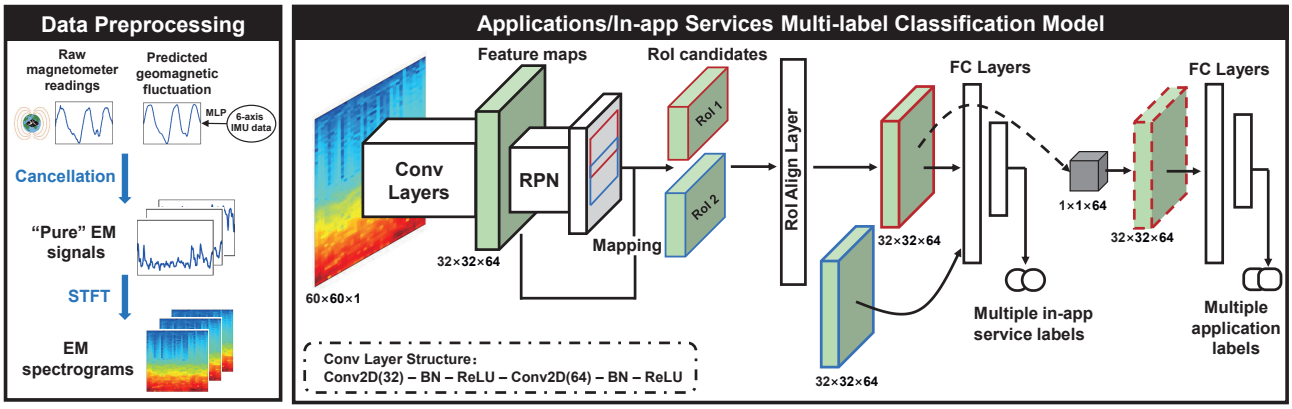


Fig. 7. Architecture and pipeline of the preprocessing and the multiple apps/in-app services classification algorithm in MAGTHIEF.

to generate feature representations for subsequent task analysis (*e.g.*, segmentation and multi-label classification). The proposed feature generation module in Fig. 7 applies two layers of Conv2D filters directly to EM spectrograms, and each layer is followed by a ReLU activation function.

2) Region proposal network for app detection: Inspired by the Faster Region-CNN [13], we developed a region proposal network (RPN) to facilitate the detection of EM features pertaining to specific apps. Using overall EM feature maps (from convolutional layers) as input, the RPN outputs a set of rectangular region proposals, each of which is assigned an objectness score.

Since the targeting app in the overall spectrogram feature maps can be at any location with arbitrary sizes, searching the whole feature maps for regions of all possible locations and sizes is computationally prohibitive. To generate region proposals, we slide the anchor boxes over the convolutional feature map to determine the region locations. Then, a RoI pooling layer takes the cropped feature map as input and outputs the unified feature map with $n \times n$ size (here $n = 3$). Each sliding box is mapped to a lower-dimensional feature, which is fed into two sibling fully-connected (FC) layers: a region-regression layer (box-reg) and a region-classification layer (box-cls). Note that at each sliding-window location, we simultaneously predict the k possible regions (also called anchor boxes) containing the app feature, which are centered at the sliding window and with k pre-fixed sizes. In this work, we use $k = 6$ aspect ratios with the same time-axis length. And the detailed design of our RPN is shown in Fig. 8.

Manually marking each app region in the feature maps after the shared convolutional layer would be impractical. Thus, to identify ground truth regions by which to train the RPN, we used an attention based model - STN network [11] to automatically learn and crop out the appropriate regions. The convolutional layer structure used for feature generation in the STN was the same as that outlined in Sec. III-C1. To account for the nine in-app service clusters, we trained nine corresponding STNs (with nine different classifiers), with the aim of determining whether the overall feature maps contain all of the in-app services and locating the corresponding

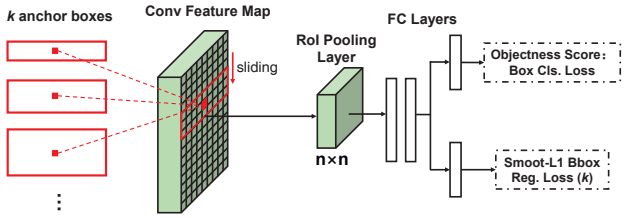


Fig. 8. Detailed structure of region proposal network.

regions for use as ground-truth regions.

After training the RPN, it outputs a set of Region of Interest (RoI) candidates that achieve objectness scores exceeding a predefined threshold (*e.g.*, 0.5). The feature map bounding boxes associated with RoI candidates vary in size, and must therefore be resized (scale-normalized) via linear interpolation in an RoI align layer [13]. The RoI candidates and their associated feature maps are fed into the multi-label classification module.

3) Multi-label in-app service/app classification: We designed a two-level classification structure for outputting both in-app service and app labels. The first level was designed as two fully-connect (FC) layers to output in-app service labels using the previously-learned in-app service feature (RoI candidates) as the input. In the second level, in-app service features are reintegrated into the feature space by mapping them to a number of channels using an additional 1×1 convolutional layer. The resulting RoI feature maps serve as inputs for another two FC layers, which outputs the app label.

4) Loss function and training: Using these definitions, we minimize an objective function following the scheme outlined in [13]. The loss function used for the overall multi-app spectrogram is as follows:

$$L(p_i, t_i, a_i, ia_i) = \sum_i \lambda_1 L_{cls}(p_i, p_i^*) + \sum_i \beta p_i^* L_{reg}(t_i, t_i^*) + \sum_i p_i^* (\lambda_2 L_{cls}(a_i, a_i^*) + \lambda_3 L_{cls}(ia_i, ia_i^*))$$

Here i is the index of a box in a mini-batch and p_i is the predicted probability of box i containing app features. Ground-truth label p_i^* is 1 if the box contains the app, and

TABLE III
THE 50 MOST POPULAR APPS FOR MOBILE DEVICES.

Category	App List (of both Android and iOS)
Music & Video	YouTube, Netflix, TED, TikTok, Spotify, Pandora, NetEase Cloud Music
Shopping	Amazon, eBay, Taobao, Jingdong
News	Yahoo, CNN, BBC, Headlines, Google(Apple) News
Social Networking	WhatsApp, Facebook, Instagram, Snapchat, WeChat, Weibo, LINE, LinkedIn
Games	Minecraft, Clash of Clans, Pokemon GO, PUBG Mobile, Honor of Kings, Hearthstone
Photography	FaceApp, PicsArt Photo Editor, Meitu
Travel&Navi	Airbnb, CTrip, Uber, Google Maps, AutoNavi(AMap)
Productivity	Gmail, Yahoo Mail, Microsoft Word, Microsoft Excel, Adobe Acrobat Reader, Evernote
Others	Microsoft Edge, Google Chrome, Alipay, Paypal

is 0 if not. Similarity, a_i and ia_i are the predicted result of app type and in-app service type, while a_i^* and ia_i^* are the corresponding ground-truth labels. The classification loss function L_{cls} utilizes cross entropy loss over classes. λ_1, λ_2, and λ_3 are weights for each individual classification cost functions, and we set λ_1 as 1, λ_2 and λ_3 as 0.5 in this study. Note that t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the ground-truth box containing app features. To derive the regression loss, we use $L_{reg}(t_i, t_i^*) = Smooth_{L1}(t_i, t_i^*)$ defined in [14], and we adopt the parameterizations of the 4 coordinates as designed in [15]. The term $p_i^* L_{reg}$ indicates the regression loss is activated only for boxes containing apps ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$). To account for varying sizes of boxes (RoIs), we learned a set of k bounding-box regressors, each of which was responsible for one aspect ratio. Note that the k regressors did not share weights. The *box-reg* are also normalized by a balancing parameter β. In our implementation, we set the $\beta = 0.01$ to weaken the importance of regression because our tasks are to obtain the app/in-app service labels instead of their exact locations on the EM spectrogram.

Training of the proposed DRCNN was implemented in two stages: pre-training and overall training. We began by pre-training the convolutional layers in the STNs to automate the locating the possible regions in terms of in-app service. We then fixed the parameters of the pretrained convolutional layers, and combined the following RPN, RPI align layer and FC layers to perform the overall training. To ensure compatibility with as many mobile devices as possible, we established two classification models – one for each of the two OS platforms (Android and iOS). We divided the training dataset in accordance with the OS, and trained two multi-label classification models using the same network structure and different weight parameters.

IV. EVALUATION

A. Experiment Setup

Dataset Collection: We recruited 12 individuals (5 females) to use mobile apps in our experiments. Each participant was tasked with arbitrarily using apps in the Table III[1], on the pre-assigned devices (see Table IV) during a period of a week. On the participant's smartphones, we installed an app tasked with continuously collecting the built-in magnetometer, 6-axis IMU data in the background at the maximum sampling rate provided by the device. While using each app, the users were required to use all of the in-app services and recorded the corresponding labels. The same clustering analysis in Sec. III-B was then used to assign new in-app service labels in accordance with the name of the nearest cluster.

TABLE IV
THE 12 SMARTPHONES USED IN THE EXPERIMENTS.

Category	Device Name (ID)
Android phones	Huawei Nexus 6P (1), Huawei P20Pro (2), Huawei Nova2 (3), Oppo Reno 4Pro (4), Samsung Galaxy S7 (5), OnePlus 7Pro (6), Xiaomi Redmi K20Pro (7), Xiaomi Mi8 (8)
iPhones	iPhone 11 (9), iPhone X (10), iPhone 8Plus (11), iPhone 7Plus (12)

Metric: The macro-averaged F1 score is utilized to evaluate the performance of the multi-label classifications on the apps and in-app services. For the app multi-label classification, we assume that p_i and r_i are the precision and recall of the ith app, the F1 score can be computed as: $F1_i = \frac{2 \times p_i \times r_i}{p_i + r_i}$. And the macro F1 score can be computed as: $F1_{macro} = \frac{1}{|L|} \sum_{i=1}^{|L|} F1_i$, where $|L|$ is the number of app classes. The same metric is utilized on the in-app service classification.

Implementation: We transferred the EM time-series data into uniform spectrograms (with 120×120 pixels) using the STFT and a image resize method. All models were initialized with learning rate of 0.001, which was further reduced after 5000 iterations. A momentum of 0.9 and weight decay of 0.0005 was used. Our model implementations were based on the tensorflow version of faster RCNN and STN with modifications, and all experiments were performed on one NVIDIA Tesla V100 GPU card.

B. Methodologies Evaluation

This sub-section examines the effectiveness of our proposed the DRCNN model on the multi-task classification of apps and in-app services.

1) Time interval vs. classification performance: MAGTHIEF was tasked with the identification of both the multiple simultaneously running apps (as well as in-app service information) from the built-in magnetometer readings over fixed time intervals of various lengths. Assigning a short time interval would obtain fine-grained app usage information; however, this would greatly hinder classification performance, due to a lack of time series information for feature extraction. After the removal of time-dependent fluctuations, we divided the magnetometer readings into segments matching the assigned time intervals to generate spectrogram images ($60 \times 60 \times 1$ pixels), and retrained the whole DRCNN models for each sampling rates and evaluated their performances. As shown in Fig. 9, setting a time interval of greater than 3

[1]We selected the 50 most popular apps on the Google Play and Apple Store

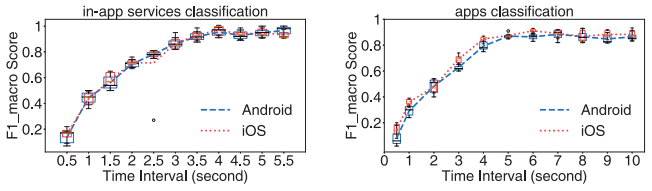
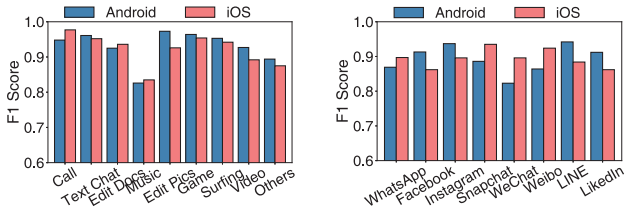


Fig. 9. Length of time interval vs. multi-label in-app services/apps classification performance.



(a) In-app services classification. (b) Apps classification.

Fig. 10. Performance of the multi-label apps/in-app services classification with the proposed DRCNN. Note that we partly list the results of apps (in social networking category) due to the space limitation.

seconds improved the macro F1 macro score of multi-label in-app services classification to roughly 0.958 with negligible standard deviation, and 5 seconds for apps classification (with 0.876 macro F1 score). In the following experiment, we set the time interval as 5 seconds for both the in-app service and app classification.

2) Multi-label in-app services classification: Fig. 10(a) showed average multi-label in-app services classification results in the terms of F1 score. As shown in the figure, our DRCNN model achieves relatively high F1 score on recognizing the multiple running in-app services. Both Android and iOS classification models have the worst performance on recognizing "Playing Music" among the nine in-app services. This is because plying music consumed little CPU recourses and involved infrequent user interaction, the corresponding EM spetrogram features were relatively weak.

3) Multi-label apps classification: Fig. 10(b) partly showed average multi-label apps (in social networking category) classification results in the terms of F1 score. The experiment results verified that the app's label can also be identified with high F1 scores, thanks to the feature transfer mechanism. Considering the thousands of app types in the market, we also sought to determine whether the increasing number of training apps would decrease the multi-label apps classification performance. As shown in Fig. 11, the identification performance of MAGTHIEF did decrease with the the number of training apps increased, but it also began relatively slowing decreasing or relatively stabilizing when the number exceeded 25, regardless of applying to Android or iOS phones.

4) Performance on unseen smartphones: We evaluated the classification performance of the proposed scheme when applied to smartphones for which the model had not previously been trained. For each Android phones in the Table IV as a testing device, we retrained the DRCNN model from scratch with the training dataset of other Android phones except itself, and evaluated the multi-label classification performance. The same process was executed for each iPhone. As shown in Fig. 12, the classifiers performed well when facing unseen

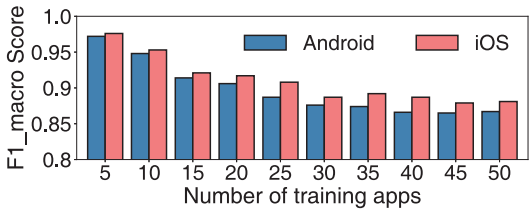


Fig. 11. Multi-label apps classification performance decreases slowly with the increase of number of training apps.

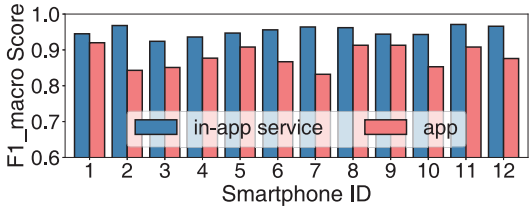


Fig. 12. Performance of our MAGTHIEF on different smartphones.

smartphones with the same operating system.

5) Comparison with baseline methods: Given the multi-label in app services/apps classification tasks, we transform them into many one-versus-the-rest binary classification subtasks. We firstly utilized the spectrogram image features (SIF [16]) and the machine learning algorithms (*e.g.*, SVM and RF) as the traditional baseline methods to handle with the multi-label classification tasks. Then, we also trained many binary CNNs/STNs as the deep learning baseline methods to compare our designed networks. Fig. 13 presents the average F1 macro scores of all multi-label in-app services/apps classification methods assessed in this study on the Android and iOS devices: Multi-SVM $(0.55/0.47)$, Multi-SVM $(0.67/0.51)$, Multi-CNN $(0.83/0.72)$, Multi-STN $(0.96/0.87)$, and our designed DRCNN$(0.95/0.87)$. The results showed the DRCNN was an outstanding classifier and could approximate the classification performance of the Multi-STN, which was used to provide ground truth regions. It is worth mentioning that the DRCNN has a more concise and single network structure than the many binary classifiers.

C. Robustness against geomagnetic noise

App classification performance with and without the proposed geomagnetic offset signal cancellation scheme (preprocessing step in Fig. 7) was evaluated in an experiment involving volunteers using mobile devices in six real-world scenarios (lying/sitting/walking in the lab and standing in the metro/bus/elevator). As shown in Fig. 14, preprocessing greatly improved the stability and accuracy of both the in-app service and app classification, particularly when the smartphones were moved frequently.

D. Influence of battery levels

The EM signals emitted by smartphones could potentially be altered by some factors, such as the battery level. We tested two smartphones and results are shown in Fig. 15, operating the devices in power saving mode ($\leqslant 20\%$ battery) had a relatively profound influence on the performance of the multi-label classification: in-app services (-0.072) and app types

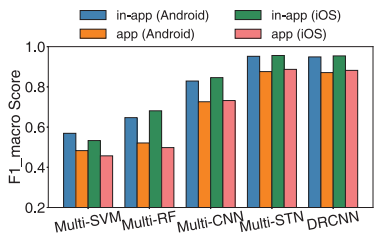


Fig. 13. Performances between different multi-label classification models.

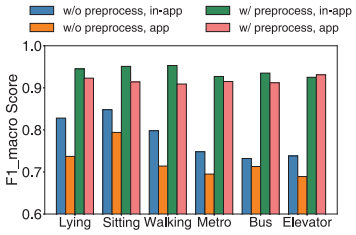


Fig. 14. Performances of the DRCNN w/ and w/o cancellation of geomagnetic noise.

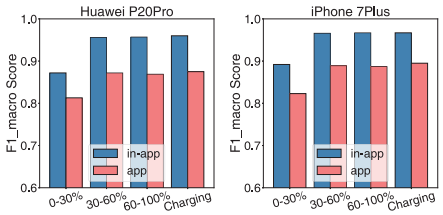


Fig. 15. Influence of battery on the performance of our MAGTHIEF.

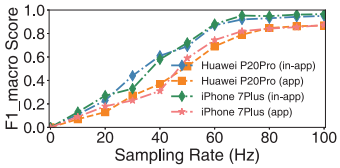


Fig. 16. Influence of sampling rate on the classification performance.

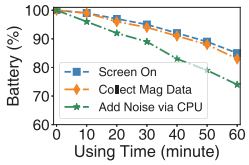


Fig. 17. Power consumption of the second countermeasure.

(-0.064). This can probably be attributed to the power-saving conditions, in which the CPU operates at a lower frequency, videos are not preloaded, games are in the low frame rate mode, and etc..

E. Overall comparison with other EM-based app classification method

In the last, we compared the proposed MAGTHIEF performance of real-time uncovering app usage information from the built-in magnetometer readings with the related EM-based app classification method MagAttack [7], [8]. Time Duration Accuracy (TDA) is adopted as the evaluation metric which evaluates the total time durations of each app that are correctly labeled, and TDA is calculated as: $TDA = \frac{T(A \cap \hat{A})}{T(A)}$, where $T(A)$ denotes the time duration of the object app usage O, and $T(A \cap \hat{A})$ is the time duration in which the app labels are correctly identified.

We let a participant to use the Huawei P20Pro at will, and recorded the app's duration time as well as the builtin magnetometer reading. The EM data analysis models in MagAttack and MAGTHIEF were utilized to identify the user's app usage information. We calculated the TDA accuracy with the app classification results. and found our MAGTHIEF achieved a TDA of 89.5%, and MagAttack[2] achieved 17.9%. The overall performance indicates that our MAGTHIEF indeed has the ability to uncover the detailed app usage information in each time duration in the actual scene.

V. DISCUSSION

Researchers have developed a number of methods to prevent the leakage of information via EM disturbance, such as physical shielding and explicit user permissions pertaining to the magnetometer sensor [8]. However, physical shielding requires modifying the hardware of existing smartphones, and imposing permissions on the magnetometer would be implausible, due to the fact that nearly all apps must sense the orientation of the device and adjust the user interface accordingly. In this study, we developed two countermeasures against our system.

[2]Note that the MagAttack assumes the user uses the app after being launched until the next app is launched.

A. Limiting sampling rates

We first sought to verify whether the sampling frequency of the built-in magnetometer would affect the classification accuracy of the MAGTHIEF system. This was achieved via downsampling and upsampling in order to alter the sampling rates in the dataset. We then retrained the MAGTHIEF models and evaluated the results. Fig. 16 lists app classification accuracy when using data obtained over a range of sampling rates. When the sampling rate was limited to $20Hz$, the macro F1 score of the system was 0.157, indicating that MAGTHIEF was unable to obtain valid information related to app usage.

B. Adding EM noise through control of the CPU

We also verify whether the CPU could be manipulated to confuse the sensor readings. We developed a custom interference app to run continuously in the background instructing the CPU to run a loop code or execute "sleep" instructions at irregular intervals [17]. When setting the run-sleep ratio as 80% in the interference app, it can result in a 0.417/0.463 decrease in macro F1-score in classifying in app services/apps respectively. We also recorded the power consumption of the MAGTHIEF app and the customized interference app. As shown in Fig. 17, running the MAGTHIEF app continuously for half an hour increased power consumption by only 6%, compared to standby with the screen on. Running the interference app for only half an hour drained the battery by 37%, which is no doubt sufficient to affect the user experience.

VI. RELATED WORK

A. Electromagnetic side channel:

In previous studies, the EM side channel was exploited to enable attacks on some mobile devices. In [18], the EM signals emitted by laptops were detected using customized antennas and a software-defined radio receiver for the extraction of RSA and ElGamal keys. [19] proposed a similar attack strategy was demonstrated at lower frequencies. In [17], EM side-channel signals were used to create a novel near-field communication system between mobile devices. In [20], electric appliances were characterized based on the EM radiation signals they emitted. In [7]–[9], [21], researchers exploited the reaction of magnetometers to EM activity to infer activities corresponding to the launching of apps or opening of browser websites. [22] also utilized EM signals to implement user fingerprinting.

In the current work, we demonstrate the feasibility of using magnetometer readings to infer the complete mobile app usage behavior, such as the multiple simultaneously running apps and the corresponding in-app service in each app.

B. Other side channels:

Side channels other than EM signals can be used to infer user behavior in the usage of mobile devices: **(i)** Several researchers have shown that power consumption traces (collected in the form of *sysfs* files [23]) are highly correlated with CPU activity patterns, and could therefore be used to infer the opening of apps [24]. However, the sampling rate for battery monitoring is generally low ($\leq 5Hz$). **(ii)** Researchers have also employed data-usage statistics (*i.e.*, tcp packages, memory footprint, browser cache) to infer user behavior [25]–[27]. Note however that accessing data-usage statistics on mobile devices via the */proc* file system can only be achieved on rooted devices. **(iii)** Researchers have demonstrated the efficacy of using built-in motion sensors to elucidate the motion of users [28]. Researchers have also utilized motion sensor data to infer when the user is tapping and gesturing during data input [29]. In contrast, the EM signals emitted by mobile devices provide valuable information pertaining to app usage and user behavior.

VII. Conclusions

This paper outlines a novel monitoring system to enable the theft of sensitive app usage information without the need for user permissions. The proposed MagThief system employs an elaborate deep region convolutional neural network for the identification of multiple apps running simultaneously and the corresponding in-app service in each app. In extensive experiments on 12 mainstream smartphones and 50 popular apps, a prototype of the MagThief system can achieve high average macro F1 scores of $0.87/0.95$ when identifying multiple apps/in-app services respectively.

Acknowledgment

We are grateful to our anonymous reviewers for their constructive feedback. This work is supported by NSFC (61936015, U1736207, 62072306), Startup Fund for Youngman Research at SJTU, and Program of Shanghai Academic Research Leader (20XD1402100).

References

[1] Z. Tu, R. Li, Y. Li, G. Wang, D. Wu, P. Hui, L. Su, and D. Jin, "Your apps give you away: distinguishing mobile users by their app usage fingerprints," *ACM UbiComp*, vol. 2, no. 3, pp. 1–23, 2018.

[2] C.-W. You, Y. Chuang, H.-Y. Lin, J.-T. Tsai, Y.-C. Huang, C.-H. Kuo, M.-C. Huang, S. J. Wu, F. W. Liu, J. Y.-J. Hsu, and H.-C. Wu, "Sobercomm: Using mobile phones to facilitate inter-family communication with alcohol-dependent patients," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, pp. 1–31.

[3] Y. Qiao, X. Zhao, J. Yang, and J. Liu, "Mobile big-data-driven rating framework: measuring the relationship between human mobility and app usage behavior," *IEEE Network*, vol. 30, no. 3, pp. 14–21, 2016.

[4] S. L. Jones, D. Ferreira, S. Hosio, J. Goncalves, and V. Kostakos, "Revisitation analysis of smartphone app use," in *UbiComp*, 2015, pp. 1197–1208.

[5] H. Pan, Y.-C. Chen, L. Yang, G. Xue, C.-W. You, and X. Ji, "mqrcode: Secure qr code using nonlinearity of spatial frequency in light," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–18.

[6] Z. Zhu, H. Pan, Y.-C. Chen, X. Ji, F. Zhang, and C.-W. You, "Magattack: remote app sensing with your phone," in *ACM UbiComp Poster*, 2016, pp. 241–244.

[7] Y. Cheng, X. Ji, W. Xu, H. Pan, Z. Zhu, C.-W. You, Y.-C. Chen, and L. Qiu, "Magattack: Guessing application launching and operation via smartphone," in *ACM AsiaCCS*, 2019, pp. 283–294.

[8] N. Matyunin, Y. Wang, T. Arul, K. Kullmann, J. Szefer, and S. Katzenbeisser, "Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting," in *ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 135–149.

[9] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu, "Deepmag: Sniffing mobile apps in magnetic field through deep convolutional neural networks," in *IEEE PerCom*, 2018, pp. 1–10.

[10] Y. Fu, J. Liu, X. Li, and H. Xiong, "A multi-label multi-view learning framework for in-app service usage analysis," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 4, pp. 1–24, 2018.

[11] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *arXiv preprint*, 2015.

[12] X. Jin and J. Han, "K-means clustering," *Encyclopedia of Machine Learning and Data Mining*, pp. 695–697, 2017.

[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, pp. 1137–1149, 2015.

[14] R. Girshick, "Fast r-cnn," in *IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *IEEE CVPR*, 2014, pp. 580–587.

[16] J. Dennis, H. D. Tran, and H. Li, "Spectrogram image feature for sound event classification in mismatched conditions," *IEEE signal processing letters*, vol. 18, no. 2, pp. 130–133, 2010.

[17] H. Pan, Y.-C. Chen, G. Xue, and X. Ji, "Magnecomm: Magnetometer-based near-field communication," in *ACM MobiCom*, 2017, p. 167–179.

[18] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side-channel(s)," *CHES*, pp. 29—45, 2002.

[19] J. Longo, E. De Mulder, D. Page, and M. Tunstall, "Soc it to em: electromagnetic side-channel attacks on a complex system-on-chip," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 620–640.

[20] E. J. Wang, T.-J. Lee, A. Mariakakis, M. Goel, S. Gupta, and S. N. Patel, "Magnifisense: Inferring device interaction using wrist-worn passive magneto-inductive sensors," in *UbiComp*. ACM, 2015, pp. 15–26.

[21] Y. Cheng, X. Ji, J. Zhang, W. Xu, and Y.-C. Chen, "Demicpu: Device fingerprinting with magnetic signals radiated by cpu," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1149–1170.

[22] L. Yang, Y.-C. Chen, H. Pan, D. Ding, G. Xue, L. Kong, J. Yu, and M. Li, "Magprint: Deep learning based user fingerprinting using electromagnetic signals," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 696–705.

[23] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to peep-all: Inference attacks by malicious batteries on mobile devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 141–158, 2018.

[24] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," in *IEEE INFOCOM*, 2017.

[25] N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang, "Leave me alone: App-level protection against runtime information gathering on android," in *IEEE Symposium on Security and Privacy*, 2015.

[26] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *ACM SIGSAC*, 2013.

[27] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *IEEE Symposium on Security and Privacy*, 2012.

[28] C. Liu, L. Zhang, Z. Liu, K. Liu, X. Li, and Y. Liu, "Lasagna: Towards deep hierarchical understanding and searching over mobile sensing data," in *ACM MobiCom*, 2016, pp. 334–347.

[29] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." *HotSec*, 2011.