POWERFUL: Mobile App Fingerprinting via Power Analysis

Yimin Chen*, Xiaocong Jin*, Jingchao Sun*, Rui Zhang†, and Yanchao Zhang*
*School of Electrical, Computer and Energy Engineering (ECEE), Arizona State University
†Department of Computer and Information Sciences, University of Delaware
{ymchen, xiaocong.jin, jcsun}@asu.edu, ruizhang@udel.edu, yczhang@asu.edu

***Abstract*—Which apps a mobile user has and how they are used can disclose significant private information about the user. In this paper, we present the design and evaluation of POWERFUL, a new attack which can fingerprint sensitive mobile apps (or infer sensitive app usage) by analyzing the power consumption profiles on Android devices. POWERFUL works on the observation that distinct apps and their different usage patterns all lead to distinguishable power consumption profiles. Since the power profiles on Android devices require no permission to access, POWERFUL is very difficult to detect and can pose a serious threat against user privacy. Extensive experiments involving popular and sensitive apps in Google Play Store show that POWERFUL can identify the app used at any particular time with accuracy up to 92.9%, demonstrating the feasibility of POWERFUL.**

I. Introduction

The popularity of mobile devices has driven the fast development of attractive mobile apps, which in turn further accelerates the ubiquity of mobile devices. For example, a recent Nielsen analysis [1] found that U.S. smartphone users accessed 26.7 apps on average and spent 37 hours and 28 minutes per month in Q4 2014. Mobile app fingerprinting, by which one can know *the apps a user has installed and how s/he uses these apps*, can be used for user profiling and inferring sensitive information about the user such as hobbies, health conditions, locations, habits, and life styles. The disclosure of such sensitive information endangers user privacy.

How could the app usage information be collected? Mobile app stores such as Google Play Store and Apple App Store are obviously in the best position to collect such sensitive information. Such app stores are fortunately operated by trustworthy business giants and not a major threat against user privacy. Mobile malware can play the main role in collecting sensitive app usage information. According to Alcatel-Lucent's Motive Security Labs [2], the malware infection rate on mobile devices rose to 0.75% in Q2 2015 from 0.68% in December 2014, and there were as many Android devices infected with malware as Windows laptops in the second half of 2014 alone. Mobile malware can be embedded into apps purposefully by malicious app developers or through hacked app development tools. An instance for the later case is the XcodeGhost malware found in September 2015 and from a malicious version of Xcode, Apple's official tool for developing iOS and OS X apps. Another instance is the backdoor in Baidu Android SDK which was found in November 2015 and may have put 100 million Android devices at risk. Besides of malware-infected apps, an enterprise app may collect its employees' app usage information without prior consent.

Significant effort has been made to infer sensitive user information on mobile devices. For example, internal sensors on a mobile device have been exploited to infer user inputs on the touchscreen [3]–[7] and user locations [8], [9]. Android public resources that can be accessed without requiring user permission have also been used to infer sensitive user information in [10]–[13]. None of these schemes aims at app usage information. Existing work on mobile app fingerprinting mostly relies on traffic analysis [14]–[19], all of which require the attacker to obtain the entire web traffic from the victim's device. As a result, the attacker needs to either be in the vicinity of the victim or even compromise network service providers to obtain the traffic data, which limits their applicability. In addition, these traffic-based methods do not work well with apps which generate only a limited amount of traffic or stay offline for most of the time.

In this paper, we present the design and evaluation of POWERFUL, a novel and practical attack framework for mobile app fingerprinting on Android devices through power profile analysis. POWERFUL is built upon the observation that different apps use different components of a device (e.g., touchscreen, CPU, Wi-Fi, and Bluetooth) and have different usage patterns, which result in distinguishable power consumption profiles. POWERFUL exploits the inherent heterogeneity of app power profiles for app characterization and usage inference. Since the power profiles on Android devices can be directly accessed without requiring user permission, POWERFUL is very difficult to detect and thus poses a serious and realistic threat against user privacy. Compared with existing traffic-based app fingerprinting techniques, POWERFUL does not require the adversary to be in the vicinity of the victim or compromise network service providers. Instead, it exploits the zero-permission Android public resources to obtain the power profiles of a device for app fingerprinting. Meanwhile, POWERFUL works well with apps generating little traffic.

We summarize our contributions in this paper as follows.

- We propose POWERFUL, the first mobile app fingerprinting framework for Android devices based on power analysis. Combining signal processing and machine learning techniques, POWERFUL is able to identify the app being used from a set of candidate apps with a high accuracy based on the corresponding power profile. Since Android requires no user permission to access and collect power profiles, POWERFUL poses a serious and realistic threat against user privacy.
- We evaluate the efficacy of POWERFUL via extensive experiments on a set of 22 most popular and privacy-related apps in Google Play Store. Experiment results show that POWERFUL can identify the app being used at any particular time from a set of candidate

apps with accuracy up to 92.9% and is resilient to the change in various factors, such as locations (office, apartments, etc.), user activities (static or walking), and user variation.

The rest of the paper is structured as follows. Section II introduces the necessary background of POWERFUL, a feasibility study, and our adversary model. Section III details the components of POWERFUL. Section IV shows the design and results of our experiments. Section V summarizes the related work. Section VI concludes this paper.

II. Preliminaries

A. Background

In this section, we briefly introduce the background of Andriod's public resources. Android makes a subset of resources publicly accessible to all apps without requiring them to explicitly obtain permissions, as sharing these resources is generally considered harmless and makes them convenient to access by all apps whenever needed. The public directories in the Linux layer are an important category of the publicly accessible resources, most of which reside in two virtual filesystems: the proc filesystem (`/proc`) and the sys filesystem (`/sys`). In `/proc`, an app can access the resource usage of a process such as its usage of memory, CPU, and network, while in `/sys`, an app can find information about various kernel subsystems, hardware devices, and associated device drivers, etc. We obtain the device's voltage and current measurements from the `voltage_now` and `current_now` files, respectively, both of which are public resources residing under the `/sys/class/power_supply/battery` folder.

B. Feasibility study

In this section, we show the feasibility of inferring app usage by analyzing their power profiles. As mentioned in Section I, POWERFUL explores the distinct characteristics in the power profiles of different apps caused by the heterogeneity of their resource usage and usage patterns.

Fig. 1(a) shows the power profiles of Bank of America (BoA) and YouTube apps. We can see from Fig. 1(a) that the two apps have similar power fluctuations, but YouTube has larger minimum and maximum powers than BoA. Therefore, minimum and maximum powers can be used to distinguish the two apps. Similarly, Fig. 1(b) shows that Netflix and Skype apps exhibit distinct characteristics of power profiles in terms of minimum power, maximum power, and power fluctuations, making them distinguishable by examining these features. Moreover, Facebook and Medscape apps have the similar minimum power as shown in Fig. 1(c), but Facebook tends to have larger power fluctuations.

C. Adversary model

We assume that the attacker runs a malicious app on the victim's device. As a standard assumption in Android security literature, it is backed up by the recent report that one out of ten Android apps are affected with malware and viruses [20]. The attacker also needs to know the device model and OS of the victim's device because these two factors directly affect the power consumption of the device. Such information can be obtained from the `System` and `Build` class[1] without requesting any user permission.

[1]For example, `System.getProperty("os.version")` returns the OS version of the device.

Categories	Apps
Communication	Gmail (GM), Messenger (MSG), Skype (SKY)
Education	TED (TED)
Entertainment	Netflix (NF), YouTube (YT)
Finance	Bank of America (BoA), Chase (CHA)
Games	Candy Crush (CCS), Pokémon Go (PM)
Health & Fitness	iTriage (iT), MedScape (MED), mySugr Diabetes Logbook (SDL)
Music & Audio	Spotify (SP)
News & Magazines	CNN (CNN)
Shopping	Amazon (AM), eBay (eBay), Groupon (GR)
Social	Facebook (FB), Twitter (TW), Tinder (TD)
Travel & Local	Priceline (PL)

TABLE I: Apps and their abbreviations.

The malicious app tries to be as stealthy as possible by running in the background to escape visual detection. According to our experiments, our "malicious" app for this research has a relatively stable power consumption of less than 20 mW, which has negligible influence on the collected power profiles. The app collects the power profile of the mobile device either periodically or following a predefined schedule. The app also needs to send the collected data to the attacker in a stealthy manner, which can be easily accomplished based on existing methods. For example, the malicious app may be inserted into or collude with another app which is legitimately given the `INTERNET` permission, and this approach is adopted by most existing work such as [3]–[5]. Alternatively, the malicious app can smuggle out the data across the Internet without requiring the user permission by using intent `URI ACTION_VIEW` to open a browser and sneaking the data to the parameters of an HTTP `GET` from the receiver side [21].

D. Targeted sensitive apps

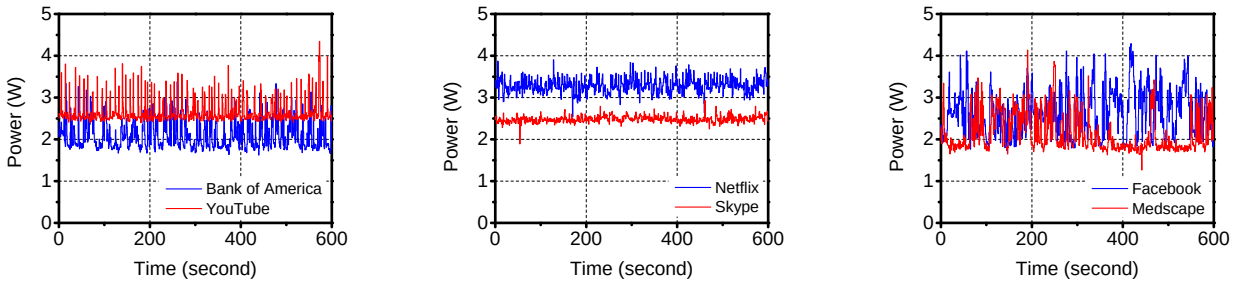
Similar to [13], [14], [18], [19], we assume that the attacker is interested in fingerprinting a small set of selected apps that are popular, highly sensitive, or contain significant private user information. This is a common assumption as it is impractical for the attacker to build a classifier for all the existing mobile apps due to the large quantity. In addition, mobile apps are constantly evolving from time to time, and the cost to build and maintain a comprehensive database would be prohibitive. Moreover, many apps contain little private information about the user, which the attacker may lack incentives to fingerprint.

Table I lists the 22 apps which we study in this paper and are selected based on the following criteria.

1. A selected app is popular and has been downloaded for more than 500,000 times in Google Play Store.
2. A selected app is usually closely related to user privacy in some way, or its usage can be exploited by the attacker for more advanced attacks [13]. For example, communication, finance, health/fitness, shopping, and social apps can directly reveal important private information about the user, such as her/his accounts, health conditions, and online history, while location-based apps like Pokémon Go and Priceline can disclose user location traces.
3. The selected apps cover as diverse categories as possible in Google Play Store.

III. Design of POWERFUL

In this section, we detail the design of POWERFUL.



(a) Bank of America and YouTube. (b) Netflix and Skype. (c) Facebook and Medscape.

Fig. 1: Power profiles of several exemplary apps.

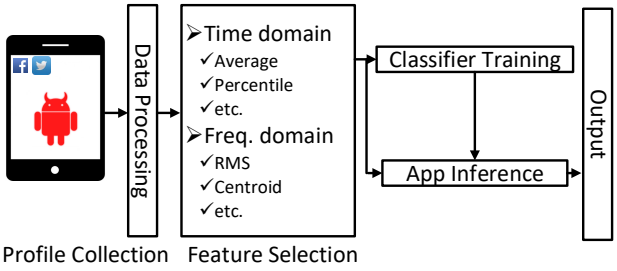


Fig. 2: Flow chart of POWERFUL.

A. Overview

As shown in Fig. 2, POWERFUL consists of the following five steps.

1. *Power profile collection*. In this step, we implement an Android app to collect the instantaneous current and voltage measurements of a Google Nexus 7 tablet when the user is using a target app. According to our experiments, our data collection app has a relatively stable power consumption of less than 20 mW and therefore has little influence on the collected power profiles.
2. *Data processing*. In this step, we process the collected power profiles by compensating the difference of power consumption due to different brightness levels and then extracting the minimums and maximums of the power profile to facilitate subsequent feature extraction.
3. *Feature extraction*. In this step, we extract a feature vector comprising features in both time and frequency domains.
4. *Classifier training*. In this step, we first obtain the power profiles of the targeted sensitive apps in Table I and use lightweight machine learning algorithms to train classifiers for subsequent testing.
5. *App inference*. In this step, given an instance of the power profile of the user's device, we use the trained classifiers to determine the app(s) being used.

B. Power profile collection

On the victim side, the malicious app collects instantaneous current and voltage measurements of the device by reading `/sys/class/power_supply/battery/current_now` and `/sys/class/power_supply/battery/voltage_now`, respectively, either periodically or following a predefined schedule. In our experiments, we set the sampling frequency to 2 Hz to strike a good balance between profile accuracy and the amount of data that need be stealthily transmitted to the attacker through the Internet. After collecting voltage and current measurements for a sufficiently long period, the app constructs the power profile that comprises a sequence of instantaneous power measurements computed as the products of the corresponding current and voltage measurements. The app also obtains the current brightness level of the device in the public system setting `android.provider.Settings.System.SCREEN_BRIGHTNESS`, which requires no user permission to access. The app finally sends the power profile and the brightness level of the victim's device to the attacker.

The attacker also builds a power profile for each target app. In particular, the attacker employs multiple users to use every target app on a device with the same model and OS and builds a power profile for each target app for subsequent classifier training and app inference.

C. Data processing

In this step, we process the raw power profiles to facilitate subsequent feature extraction. Without loss of generality, we consider a power profile $P = (p_1, \ldots, p_n)$, where p_i is the ith power measurement for all $i \in [1, n]$ and n is the total number of power measurements.

We first apply a sliding window of length W and offset factor r on P to generate a sequence of power profile samples $S_1, \ldots, S_k$ of equal length, where

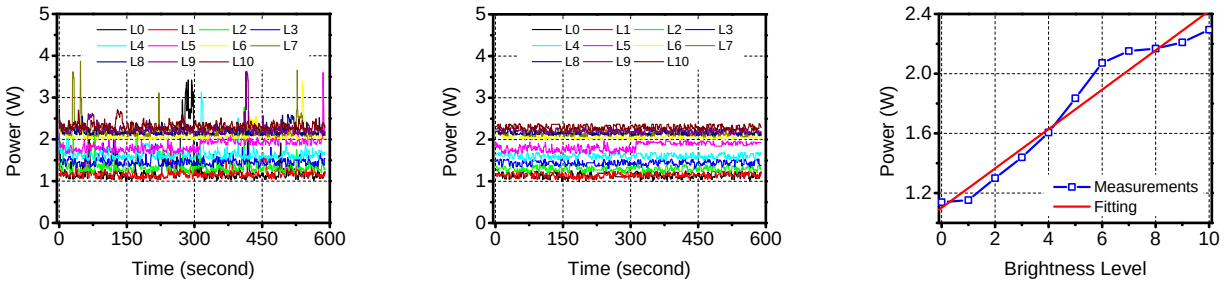
$$S_i = (p_{(i-1)rW+1}, \ldots, p_{(i-1)rW+W}),$$

for all $i = 1, \ldots, k$, and $k = \lfloor \frac{n-W}{rW} \rfloor$. In this paper, we empirically set r to 0.1 and choose W such that $rW \in \mathbb{Z}$.

For each sample S_i, we proceed with the following two steps: *power adjustment* and *min-max search*.

1) Power adjustment: In this step, we compensate the difference in power consumption caused by different brightness levels. Such adjustment is necessary because the touchscreen is a major energy-consuming component in modern mobile devices, and different brightness levels result in different power consumption rates of the touchscreen and therefore different power profiles for the same device.

Power adjustment requires a power model to characterize the relationship between touchscreen power consumption and brightness level. While several models have been proposed in the literature [22], [23], they are either device-specific due to the technology and hardware difference or require user permission to acquire the status of different components. In



(a) Original power profiles. (b) Power profiles after removing peaks. (c) Our linear power model.

Fig. 3: Power adjustment for different touchscreen brightness levels.

this paper, we adopt a simple linear relationship between the power consumption and the brightness level built from fitting empirical data.

Specifically, we collect the power measurements at different brightness levels using a modified version of our app. No third-party app is installed on the device to minimize potential impact on accuracy. The app automatically sets the brightness coefficient from 0 to 1 with an interval of 0.1 (corresponding to level 0 to level 10) and records the voltage and current measurements at a frequency of 2 Hz for 10 minutes before changing to the next brightness level. Fig. 3(a) shows the power measurements at different brightness levels. We can see that the curves exhibit multiple peaks while they are expected to be generally stable. We conjecture that the peaks are most likely due to pre-installed system apps (e.g., Google Play services) running in the background and thus should be excluded during parameter fitting. Our measurement results are similar to those collected using the tools in [22].

We remove these peaks in two steps. First, we calculate the cumulative distribution function (CDF) of the power measurements. Second, we remove the measurements above a certain percentile of the CDF, where we empirically choose 80% as the threshold. We plot the power measurements after removing the peaks in Fig. 3(b), where we can see that the resulting power profiles are generally stable.

We then calculate the average of the (approximate) 10-minute measurements as the device's power consumption rate at the corresponding brightness level. Given a set of brightness levels and corresponding power consumption rates, we further calculate the slope s and intercept b of the linear model through least-squares fitting. We plot the measurements and the fitted model in Fig. 3(c) and show the fitted parameters in Table II.

We finally adjust the power measurements using the power model obtained above. Specifically, given the device's current brightness level L, we calculate the power consumption rate difference between level L and level 0 as sL, where s is the slope of the linear power model. Then for every power profile sample $S_i = (p_{(i-1)rW+1}, \ldots, p_{((i-1)r+1)W})$, we compute a new sample $S'_i = (p'_{(i-1)rW+1}, \ldots, p'_{((i-1)r+1)W})$, where $p'_j = p_j - sL$ for all $j = (i-1)rW+1, \ldots, (i-1)rW+W$.

Parameter	Value	Standard Error
Intercept, b	1.10	0.057
Slope, s	0.132	9.65×10^{-3}

TABLE II: Fitted parameters of our linear power model.

2) Min-max search: Next, we extract the "skeleton" of each power profile sample by finding the local minimums and maximums of its power measurements.

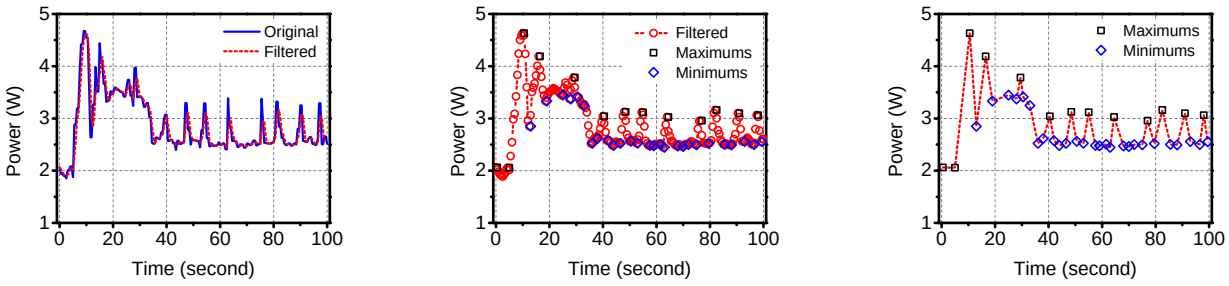
Without loss of generality, we consider a power profile sample $S' = (p'_1, \ldots, p'_W)$ and use the example in Fig. 4 to illustrate how local minimums and maximums are determined. First, we apply a five-point simple moving average (SMA) filter to smooth the power profile sample to reduce the impact of small fluctuations. Fig. 4(a) shows the original and filtered power traces in our example. Denote the power trace after filtering by $\tilde{p}_1, \ldots, \tilde{p}_W$. Then for each $\tilde{p}_j, j = 1, \ldots, W$, we select $\tilde{p}_j$ as a *local minimum (maximum)* if the following two conditions are satisfied.

- **Condition 1**: Its value is no larger (smaller) its two neighboring values, i.e., $\tilde{p}_j \leq \tilde{p}_{j-1}$ and $\tilde{p}_j \leq \tilde{p}_{j+1}$ ($\tilde{p}_j \geq \tilde{p}_{j-1}$ and $\tilde{p}_j \geq \tilde{p}_{j+1}$).
- **Condition 2**: Its value is at least δ_t smaller (larger) than the previous closest local maximum (minimum).

Here δ_t is an important parameter that need be chosen carefully. On the one hand, if δ_t is too small, too many minimums or maximums will be selected due to small fluctuations in the power profile caused by spontaneous noise, which do not contribute to the characterization of the app. On the other hand, an overly large δ_t makes the second condition difficult to satisfy, resulting in some meaningful minimums or maximums being omitted and thus poor characterization of the app. Moreover, since different apps exhibit distinct characteristics, the choice of δ_t should not be universal but app-specific. We observe that a proper δ_t should be positively correlated with the standard deviation σ of the power measurements for a given app. Therefore, we choose $\delta_t = c\sigma$ and empirically set $c = 1$ in this work. Fig. 4(b) and 4(c) show the labeled local minimums and maximums of the given power trace in Fig. 4(a). The figures shows that they capture the overall shape of the power trace, indicating the capability of extracting the "skeleton".

After finding all the local minimums and maximums from the power profile sample, we generate a vector of pair $V = \big((m_1, t_1), \ldots, (m_e, t_e)\big)$, where m_j and t_j are the jth local maximum or minimum and the corresponding time stamp, respectively, and e is the total number of local maximums and minimums. Moreover, we generate a label vector $L = (l_1, \ldots, l_e)$, where $l_j = -1$ if m_j is a local minimum and 1 otherwise. It follows that $l_j l_{j+1} = -1$ if m_j and m_{j+1} are a pair of adjacent minimum and maximum.

We further compute a power difference vector ΔV, a time difference vector ΔT, and a slope vector R from V and L using Algorithm 1, which capture the power and time differences of pairs of adjacent minimum and maximum and



(a) Effect of SMA filtering.

(b) Local maximums and minimums found by our approach.

(c) "Skeleton" composed of local maximums and minimums.

Fig. 4: Illustration of min-max search.

Algorithm 1: Computing $\Delta V, \Delta T$, and R

input : V, L
output: $\Delta V, \Delta T, R$

Initialize $\Delta V, \Delta T$, and R as empty vectors;
for $j = 1, \ldots, e-1$ **do**
  if $l_j l_{j+1} = -1$ **then**
    Expand ΔV by adding a dimension and new element $|m_{j+1} - m_j|$;
    Expand ΔT by adding a new element $t_{j+1} - t_j$;
    Expand R by adding a new element $\frac{\Delta V(i)}{\Delta T(i)}$;
  return $\Delta V, \Delta T$, and R;

the sharpness of the corresponding rising or falling slopes, respectively.

D. Feature extraction

In this step, we extract features from both time and frequency domains to represent a given power profile sample. For each power profile sample, the extracted features form a vector, which is referred to as an instance hereafter.

1) Features in time domain: For a given power profile sample $S' = (p'_1, \ldots, p'_W)$, we extract the following statistic measures as the features in time domain.

- The average, the 20th, 50th, and 80th percentile, the standard deviation (SD), the maximum, and the minimum of $(p'_1, \ldots, p'_W)$, denoted by $p_{avg}, p_{20pctl}, p_{50pctl}, p_{80pctl}, p_{SD}, p_{max}$, and p_{min}, respectively.
- The average, the 20th, 50th, and 80th percentile, SD of $\Delta V, \Delta T$, and R, respectively. We denote them as $\Delta V_{avg}, \Delta V_{20pctl}, \Delta V_{50pctl}, \Delta V_{80pctl}, \Delta V_{SD}, \Delta T_{avg}, \Delta T_{20pctl}, \Delta T_{50pctl}, \Delta T_{80pctl}, \Delta T_{SD}, R_{avg}, R_{20pctl}, R_{50pctl}, R_{80pctl}$, and R_{SD}, respectively.

2) Features in frequency domain: Given a power profile sample $S' = (p'_1, \ldots, p'_W)$, we first calculate its Fourier Transform as $Q = (q_1, \ldots, q_W)$ using Fast Fourier Transform (FFT). We then extract the following features from Q.

- *Root-mean-square (RMS) energy*. The RMS energy is an approximation of the average signal strength and calculated as the square root of the arithmetic mean of the squares of Q.

$$\text{RMS} = \sqrt{\frac{1}{W}\sum_{k=1}^{W} q_k^2}.$$

- *Spectral centroid*. The spectral centroid represents the "center" of Q and is the weighted mean of the frequencies of Q with q_k as the weights.

$$\mu = \frac{\sum_{k=1}^{W} q_k f_k}{\sum_{k=1}^{W} q_k},$$

where $f_k = \frac{k f_s}{2}$, and f_s is the sampling frequency.

- *Spectral entropy*. The spectral entropy captures the locations of the peaks of Q and is computed as

$$H = \sum_{k=1}^{W} \omega_k \log_2 \omega_k,$$

where $\omega_k = \frac{q_k}{\sum_{k=1}^{W} q_k}$ is the normalized frequency.

- *Spectral irregularity*. The spectral irregularity captures the jitter or noise in Q and is given by

$$\text{irregularity} = \frac{\sum_{l=1}^{n_p} (\phi(x) - \phi(x+1))^2}{\sum_{x=1}^{n_p} \phi(x)^2},$$

where $\phi(x), x = 1, 2, \ldots, n_p$ are the peaks in Q and $\phi(n_p + 1) = 0$.

- *Spectral spread*. The spectral spread is to capture the dispersion of Q with respect to its centroid and calculated as the standard deviation of the spectral distribution.

$$\rho = \sqrt{\sum_{k=1}^{W} [\omega_k (f_k - \mu)^2]}.$$

- *Spectral skewness*. The spectral skewness captures the symmetry of Q and is computed as

$$\text{skewness} = \frac{\sum_{k=1}^{W} \omega_k (f_k - \mu)^3}{\rho^3}.$$

- *Spectral kurtosis*. The spectral kurtosis captures the flatness of Q compared with Gaussian distribution and is given by

$$\text{kurtosis} = \frac{\sum_{k=1}^{W} \omega_k (f_k - \mu)^4}{\rho^4}.$$

- *Spectral flatness*. The spectral flatness captures how energy is spread across Q and is given by

$$\text{flatness} = \frac{(\prod_{k=1}^{W} q_k)^{\frac{1}{W}}}{\frac{1}{W}\sum_{k=1}^{W} q_k}.$$

E. Classifier training

In this step, we train a classifier from a training set with known labels. POWERFUL can work with many existing machine learning techniques. In this paper, we consider three lightweight supervised machine learning techniques, i.e., C4.5, random forest (RF), and support vector machine (SVM) for classifier training and testing. In particular, C4.5 generates a decision tree according to C4.5 algorithm and uses the decision tree to map an instance to a finite set of values, which are the class labels [24]. RF builds a forest of uncorrelated decision tress for classification [25]. The unique feature of RF is that it keeps selecting a random subset of features to control the variance of classification result during the process of generating the forest. SVM maps the instances of different classes in space such that they are divided by a clear gap which is as wide as possible [26], [27]. In Weka [28], the corresponding implementations of the three techniques are `J48` , `RandomForest`, and `LibSVM` class, respectively.

F. App inference

In this step, we infer the app being used from the given power profile. Specifically, the power profile first goes through Data Processing and Feature Extraction steps and becomes a series of instances. Then given a specific instance, we use the classifier trained in Section III-E to calculate the class label. The app corresponding to the output class label is considered as the app being used at the particular time.

IV. Performance Evaluation

In this section, we first describe the experimental setup, then introduce the performance metric adopted, and finally report the evaluation results in details.

A. Experiment setup

1) Data collection: We used a Google Nexus 7 tablet with Android 4.4.4 and a Google Nexus 6 smartphone with Android 5.1.1 with our app installed to collect power profiles. We recruited 24 users to participant in the experiments, including two females and twenty-two males. Each participant was assigned ten apps and then delivered the following instructions. First, each participant adjusted the screen brightness level according to her/his need, connected the device to a reliable Wi-Fi AP, and turned off the Bluetooth connection. Then, s/he activated the data collection app and started to use one of the assigned apps. Each participant was required to stay where s/he was, such as in the office or at her/his apartment, when s/he was using an app. Each participant was also asked to use only one assigned app at any given time. For each assigned app, the participant was asked to use it for at least half an hour. To complete the data collection of one assigned app, the participant was allowed to use it either continuously or from time to time, as long as the total usage time of each assigned app exceeded half an hour. We also required the participant not to change the brightness setting during the experiments.

2) Evaluation protocol: Our main dataset, denoted by $\mathcal{S}$, consists of the instances of all 22 sensitive mobile apps. In total, 24 participants are involved in the experiments, who are graduate students in Arizona State University and age between 20 and 35. Each participant was assigned ten apps to use. Considering the total number of participants in the experiments, only two apps (BoA and CHA) are used by ten participants while all the other apps are each used by eleven participants.

For each run of evaluation, we randomly divided $\mathcal{S}$ into one training set $\mathcal{S}_{\text{train}}$ and one testing set $\mathcal{S}_{\text{test}}$. Based on our adversary model, $\mathcal{S}_{\text{train}}$ is built by the attacker while $\mathcal{S}_{\text{test}}$ is obtained from the malicious app on the victim's device. For each app we studied, we randomly selected one participant as the victim and allocated her/his instances to $\mathcal{S}_{\text{test}}$ while allocating the remaining instances of the same app to $\mathcal{S}_{\text{train}}$. By doing so, we ensured that the instances of $\mathcal{S}_{\text{train}}$ were from the attacker and those of $\mathcal{S}_{\text{test}}$ were from the victim. Finally, we ran the evaluation for 40 times and reported the average results.

B. Performance metric

In this paper, we use *identification rate* as the performance metric to evaluate the attack capability of POWERFUL. In specific, for each app we study, we define the identification rate as the ratio between the number of correctly-classified instances and that of all instances of the app in a testing set. A higher identification rate means that given a power profile, the attacker (POWERFUL) is able to identify the corresponding app used by the victim more accurately, thus posing a more serious threat on user privacy.

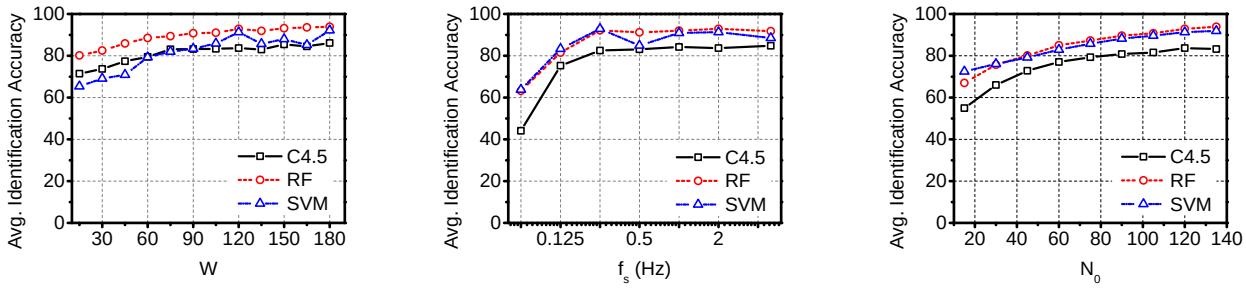
C. Experimental results

1) Impact of window length: Fig. 5(a) shows the average identification rate of all the apps when window length W increases from 15 to 180. As we can see, the average identification rate increases as W increases for all three machine learning techniques. This is expected because a larger W means that the power profile with more measurements is used for identification, and it is thus more likely to extract app-specific features to increase the identification rate. On the other hand, a smaller W means that the attacker only needs to collect a power profile for a shorter period, making the attack more practical. We set W to 120 for the rest of our experiments, corresponding to a duration of 60 seconds.

2) Impact of sampling frequency: Fig. 5(b) shows the average identification rate across all the 22 apps with sampling frequency f_s varying from 0.00625 Hz to 5 Hz. We can see that the average identification rate increases as f_s increases for all three machine learning techniques. The reason is that the higher f_s, the finer-grained characteristics of the collected power profiles. We can also observe that the identification rate tends to be stable when f_s is higher than 2 Hz. This is mainly because that the extracted features do not change much when further increasing f_s. Since a higher f_s leads to more power profiles that need be stealthily transmitted over the Internet, making our attack easier to detect, f_s is set as 2 Hz in our experiments to strike a balance.

3) Impact of number of training instances: Fig. 5(c) shows the average identification rate across all the 22 apps with the number of training instances N_0 varying from 15 to 135. We can see that the average identification rate increases as N_0 increases for all three machine learning techniques. This is expected because the classifiers get better trained with more training instances and consequently achieve higher identification rate. As a result, the attacker always uses all the available training instances in practice.

4) Feature importance: We have also studied the importance of different features used in POWERFUL, characterized by *information gain* [29], [30]. Specifically, information gain measures the amount of information about class prediction, given that the only information available is the presence of



(a) Impact of W. (b) Impact of f_s. (c) Impact of N_0.

Fig. 5: Impact of different factors on POWERFUL.

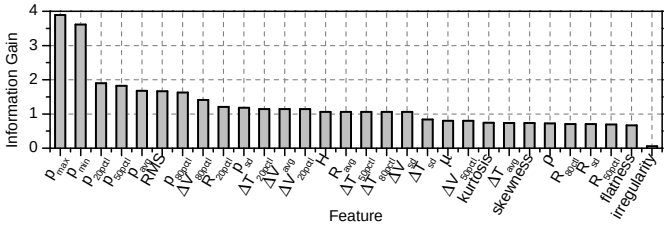


Fig. 6: Importance of features.

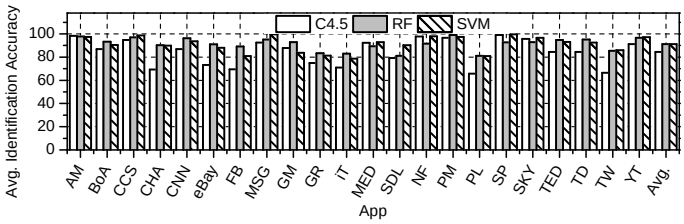


Fig. 8: Identification accuracy of POWERFUL on Nexus 6.

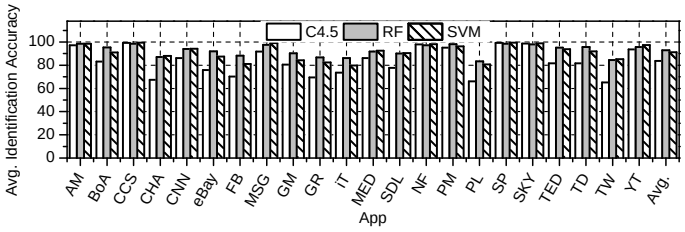


Fig. 7: Identification accuracy of POWERFUL on Nexus 7.

a feature and the corresponding class distribution. The higher information gain, the more important a feature is, and vice versa. In Weka, we obtain the information gain of different features using its `InfoGainAttributeEval` class. Fig. 6 shows the information gain of all the features of POWERFUL in descending order. We can see that two most important features are p_{max} and p_{min} while the least important is irregularity.

The less important features (i.e., with small information gain) still affect the overall classification results. For example, if we remove the features with information gain less than one, the average identification rate decreases from 92.9% to 86.1%. We therefore use all the extracted features for classifier training and testing to achieve higher identification rate.

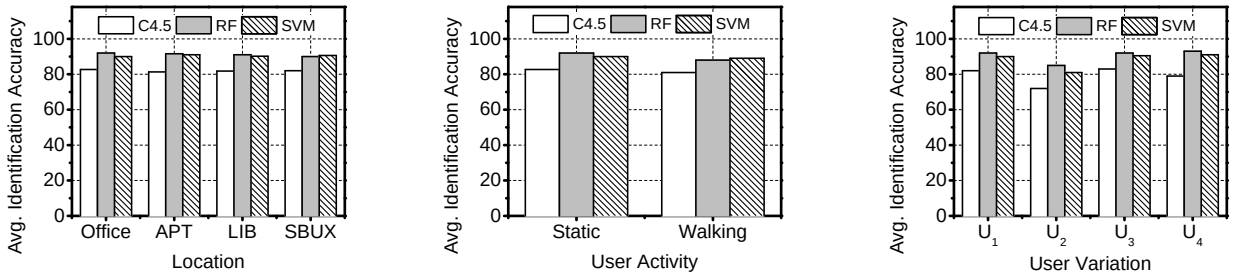
5) Attack on Nexus 7: Fig. 7 shows the identification rate of POWERFUL on a Google Nexus 7 with Android 4.4.4. As we can see, POWERFUL can correctly identify the 22 apps with high probabilities, and the identification rates using RF and SVM are similar and higher than that of using C4.5. In addition, the identification rates of most apps are higher than 80% for RF and SVM. The average identification rates of all the apps using C4.5, RF, and SVM are 83.7%, 92.9%, and 91.3%, respectively. In [19], the authors reported an overall inference accuracy of 93.96% on a smaller set of 13 apps. We believe that the performance of POWERFUL is similar to that of state-of-the-art solution.

6) Attack on Nexus 6: Fig. 8 shows the identification rate of POWERFUL on a Google Nexus 6 with Android 5.1.1. We can see that the results are similar to those on a Google Nexus 7 with Android 4.4.4. The average identification rates of all the apps using C4.5, RF, and SVM are 84.45%, 91.3%, and 91.23%, respectively. These results confirm that POWERFUL can work with devices of different models.

7) Robustness: We also conducted a separate set of experiments to evaluate the robustness of POWERFUL to locations, user activities, and user variation.

Location. To evaluate the impact of locations, we let two participants to use the apps in Table I in four different locations, including our office, their apartments (APTs), the university library (LIB), and a Starbucks (SBUX) store, where Wi-Fi access is available. Each participant used each targeted app for five minutes in the same location with a Google Nexus 7. We then applied the trained classifiers to the collected power profiles and obtained the average identification rate. Fig. **??** shows the average identification rate of POWERFUL with different locations. As we can see, the average identification rate is relatively stable across different locations. These results indicate that POWERFUL is robust to the change in location and thus can effectively fingerprint sensitive mobile apps even if the victim is at different locations. The main reason is that location has very limited impact on the power profiles and thus little impact on the classification results.

User Activity. We tested the performance of POWERFUL when users are conducting different activities. Specifically , we let two participants to use the apps in Table I while they were sitting statically in our office and walking slowly along the corridor in our office building. Under each scenario, a participant used each targeted app for five minutes with a Google Nexus 7. We then applied the trained classifiers to the collected power profiles and obtained the average identification rate. Fig. 9(b) shows the average identification rate of POWERFUL under the two user activities. As expected, we can see that the average identification rates in the two scenarios are similar to each



(a) Impact of location. (b) Impact of user activity. (c) Impact of user variation.

Fig. 9: Performance of POWERFUL under different scenarios.

other. The reason is that slow user movement does not cause much change in the power profiles and thus has little impact on the classification results. Therefore, POWERFUL is robust to slow user movement.

User Variation. We let four participants each to use each app in Table I for five minutes in our office with a Google Nexus 7 and applied the trained classifiers to the collected dataset to obtain the average identification rate. Fig. 9(c) shows the average identification rate of POWERFUL on four users. As shown in the figure, the average identification rate of U_2 is lower than those of the other three users regardless of which machine learning algorithm is used. We conjecture that the variation is caused by different users having different usage patterns for the same app. For example, some users use Skype mostly for voice call and instant messaging while other users use it mostly for video call. In practice, the attacker can alleviate the impact of user variation by collecting more diverse dataset for classifier training. Nevertheless, POWERFUL can still achieve an average identification rate of 85% on U_2 using RF.

V. Related Work

This section briefs some work closely related to POWERFUL.

A. Sensitive information inference in Android

Inferring sensitive information on Android mobile devices has received much attention in recent years. Previous work [3]–[7], [31] has shown that user input on the touchscreen, such as PIN, pattern password, user name, or even sentences, can be inferred from various onboard sensors such as accelerometer, gyroscope, microphone, or camera. In [8], [9], researchers show that accelerometer, microphone, camera, and light sensor can be used to infer target user's driving routes or locations. Some of these attacks [9] require user permissions such as `android.permission.CAMERA`, while our attack does not. Although access to sensors such as accelerometer does not require user permissions, accessing such information can be easily detected by analyzing API calls (e.g., `SensorManager.getDefaultSensor(int)`) using existing app analysis tools like [32].

Inferring sensitive information from Android's public resources has also been studied. In [10], Jana *et al.* show that the websites the user has visited and other finer-grained browsing behavior can be inferred from the memory footprint of the web browser. Zhang *et al.* show that keystroke events can be identified from the ESP data in a multi-core system [11].

In [12], Zhou *et al.* demonstrate that user's location, real identity, health conditions, and driving route can be inferred from the network usage statistics of an app and the status of public Android APIs. In addition, Chen *et al.* find that the UI state of Android device can be inferred from the memory usage of an app [13].

Compared with the above work, we work on a new attack on user privacy and make use of the power profile of Android devices, which is considered to be harmless.

B. App fingerprinting

Our work is also related to the line of research in app fingerprinting, which aims at identifying apps through traffic analysis. In [14], Stöber *et al.* show that a group of apps can be identified as a whole by analyzing 3G/UMTS data traffic. In [15], Xu *et al.* design a learning system to automatically fingerprint an app using the key-value pairs in HTTP headers, while in [16], Miskovic *et al.* tackle a similar problem by exploring the scarcity of key app-identification sources. In [17], Dai *et al.* show that an app can be identified by analyzing different HTTP requests. In [18], Verde *et al.* propose a user-fingerprinting framework using NetFlow records only, rather than the entire traffic. Recently, Wang *et al.* show that the app being used can be inferred by analyzing the overheard encrypted data using machine learning techniques [19]. Compared with this line of work, our attacker does not need to in the vicinity of the adversary or compromise large network service providers. Also, our attack is valid for apps generated very limited traffic.

C. Power analysis

There has also been some effort [22], [23], [33]–[35] in power analysis on mobile devices, which mainly focuses on understanding how power is consumed. In [22], Zhang *et al.* propose to first generate power models for device components such as CPU, LCD, and Wi-Fi and then use a function of these models to determine system-level power consumption. Similar approach is also adopted in [33], either to estimate the power consumption of an individual app or to fully understand the impact of different operating systems and hardware models. In [23], [34], Pathak *et al.* propose to use system call tracing rather than the power states of hardware components to model power usage, which improves both accuracy and granularity. In [35], Brouwers *et al.* present NEAT, a novel energy analysis toolkit for smartphones, which combines both the accuracy of a customized power measurement board and detailed system traces of hardware and software together.

In [36], Michalevsky *et al.* introduce a novel attack that reveals user locations via power analysis of the user's smart-

phone. Assuming that the distance between the smartphone and the base station greatly impacts the total power consumption, they are able to infer the user's driving routes by applying machine learning techniques.

In contrast to the above work, we study a new attack using power analysis, which poses a serious and realistic threat on user privacy.

VI. Conclusion and Future Work

In this paper, we presented the design and evaluation of POWERFUL, a novel attack framework on Android mobile device which combines power analysis and machine learning for mobile app usage inference. POWERFUL exploits the app-specific characteristics of the power profiles without requiring user permission. Our extensive experiments demonstrated that POWERFUL is able to infer the app being used at a specific time with high accuracy, thus posing a realistic and serious threat to user privacy.

As our future work, we first seek to improve the identification rate of POWERFUL by exploring additional zero-permission information in Android, such as the CPU and memory usage. We also plan to fully investigate the performance of POWERFUL under large user mobility (e.g., when the user is travelling on a bus). Finally, we aim to extend POWERFUL to iOS mobile devices such as iPhone and iPad.

Acknowledgement

This work was supported in part by the US National Science Foundation under grants CNS-1421999, CNS-1320906, CNS-1514381, CNS-1619251, CNS-1700032, and CNS-1700039. We would also like to thank anonymous reviewers for their constructive comments and helpful advice.

References

[1] http://www.nielsen.com/us/en/insights/news/2015/so-many-apps-so-much-more-time-for-entertainment.html.

[2] http://resources.alcatel-lucent.com/asset/189669.

[3] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *ACM HotMobile'12*, San Diego, CA, Feb. 2012.

[4] L. Cai and H. Chen, "On the practicality of motion based keystroke inference attack." Springer Berlin Heidelberg, Jun. 2012.

[5] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones." in *NDSS'11*, San Diego, CA, Feb. 2011.

[6] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *ACM WiSec'12*, Tucson, AZ, Apr. 2012.

[7] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion," in *USENIX HotSec'11*, San Francisco, CA, Aug. 2011.

[8] J. Han, E. Owusu, L. Nguyen, A. Perrig, and J. Zhang, "Accomplice: Location inference using accelerometers on smartphones," in *IEEE COMSNETS'12*, Bangalore, India, Jan. 2012.

[9] M. Azizyan, I. Constandache, and R. Choudhury, "Surroundsense: mobile phone localization via ambience fingerprinting," in *ACM MobiCom'09*, Beijing, China, Sep. 2009.

[10] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *IEEE S&P'12*, San Francisco, CA, May 2012.

[11] K. Zhang and X. Wang, "Peeping tom in the neighborhood: keystroke eavesdropping on multi-user systems," in *USENIX Security'09*, Montreal, Canada, Aug. 2009.

[12] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *ACM CCS'13*, berlin, Germany, Nov. 2013.

[13] Q. Chen, Z. Qian, and Z. Mao, "Peeking into your app without actually seeing it: Ui state inference and novel android attacks," in *USENIX Security'14*, San Diego, CA, Aug. 2014.

[14] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: smartphone fingerprinting via application behaviour," in *ACM WiSec'13*, Budapest, Hungary, Apr. 2013.

[15] Q. Xu, Y. Liao, S. Miskovic, Z. Mao, M. Baldi, A. Nucci, and T. Andrews, "Automatic generation of mobile app signatures from traffic observations," in *IEEE INFOCOM'15*, Hong Kong, Apr. 2015.

[16] S. Miskovic, G. Lee, Y. Liao, and M. Baldi, "Appprint: Automatic fingerprinting of mobile applications in network traffic." Springer, Mar. 2015.

[17] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: Towards automatic fingerprinting of android apps," in *IEEE INFOCOM'13*, Turin, Italy, Apr. 2013.

[18] N. Verde, G. Ateniese, E. Gabrielli, L. Mancini, and A. Spognardi, "No nat'd user left behind: Fingerprinting users behind nat from netflow records alone," in *IEEE ICDCS'14*, Madrid, Spain, Jul. 2014.

[19] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *IEEE CNS'15*, Florence, Italy, Sep. 2015.

[20] http://www.ibtimes.co.uk/android-apps-one-ten-affected-malware-viruses-states-new-research-1459576.

[21] http://www.leviathansecurity.com/blog/zero-permission-android-applications.

[22] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *ESWeek'10*, Scottsdale, AZ, Oct. 2010.

[23] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *ACM EuroSys'11*, Salzburg, Austria, Apr. 2011.

[24] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[25] T. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[26] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[27] C. Chang and C. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, Apr. 2011.

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, Nov 2009.

[29] D. Roobaert, G. Karakoulas, and N. Chawla, "Information gain, correlation and support vector machines," in *Feature Extraction*. Springer, 2006, pp. 463–470.

[30] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1997.

[31] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang, "Visible: Video-assisted keystroke inference from tablet backside motion," in *NDSS'16*, San Diego, CA, Feb. 2016.

[32] https://github.com/sonyxperiadev/ApkAnalyser.

[33] L. Ardito, G. Procaccianti, M. Torchiano, and G. Migliore, "Profiling power consumption on mobile devices," 2013.

[34] A. Pathak, Y. Hu, and M. Zhang, "Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof," in *ACM EuroSys'12*, Bern, Switzerland, Apr. 2012.

[35] N. Brouwers, M. Zuniga, and K. Langendoen, "Neat: a novel energy analysis toolkit for free-roaming smartphones," in *ACM SenSys'14*, memphis, TN, Nov. 2014.

[36] Y. Michalevsky, A. Schulman, G. Veerapandian, D. Boneh, and G. Nakibly, "Powerspy: Location tracking using mobile device power analysis," in *USENIX Security'15*, Washington, D.C., Aug. 2015.