1. Yimin Chen, Xiaocong Jin, Jingchao Sun, Rui Zhang, and Yanchao Zhang, “POWERFUL: Mobile app fingerprinting via power analysis,” IEEE International Conference on Computer Communications (INFOCOM’17), Atlanta, GA, April 2017.

**Web traffic:**

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.

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.

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.

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.

**Andriod’s public resource**:

Android makes a subset of resources publicly accessible to all apps without requiring them to explicitly obtain permissions, as sharing these resources 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 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.

Overview:

POWERFUL consists of the following five steps:

**1. Power profile collection**

We implement an Android app to collect the instantaneous current and voltage measurement of a Google Nexus 7 tablet when the user is using a target app. Our data collection app has a relatively stable power consumption of less than 20 mW and has little influence on the collected power profiles

**2. Data processing**

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**

We extract a feature vector comprising features in both time and frequency domains.

**4. Classifier training**

We first obtain the power profiles of the targeted sensitive apps and use lightweight machine learning algorithms to train classifiers for subsequent testing.

**5. App inference**

Given an instance of the power profile of the user’s device, we use the trained classifiers to determine the app(s) being used.

**Power profile collection**

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. The app also obtains the current brightness level of the device in the public system setting android.provider.Setting.System.SCREEN\_BRIGHTNESS.

**Data Processing**

We process the raw power profiles to facilitate subsequent feature extraction. We consider a power profile P = (p1, . . . , Pn) , where pi is the ith power measurement for all 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 S1, . . ., Sk of equal length.

For each sample Si, we proceed with the following two steps:

1) Power adjustment: we compensate the difference in power consumption caused by different brightness levels. We adopt a simple linear relationship between the power consumption and the brightness level built from fitting empirical data.

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.

2) Min-max search: we extract the “skeleton” of each power profile sample by finding the local minimums and maximums of its power measurements. First, we apply a five-point simple moving average (SMA) filter to smooth the power profile sample to reduce the impact of small fluctuations.

**Feature extraction**

We extract features from both time and frequency domains to represent a given power profile sample.

1) Features in time domain: For a given power profile sample S’ = (p1’, … pw’), the average, the 20th, 50th, and 80th percentile.

2) Features in frequency domain: we first calculate its Fourier Transform as Q = (q1, … , qw) using Fast Fourier Transform (FFT). We then extract the following features from Q, Root-mean-square (RMS) energy, Spectral centroid, Spectral entropy, Spectral irregularity, Spectral spread, Spectral skewness, Spectral kurtosis, Spectral flatness.

**Classifier training**

We train a classifier from a training set with known labels. We consider three lightweight supervised machine learning techniques, C4.5, random forest (RF), and support vector machine (SVM) for classifier training and testing.

**App inference**

We infer the app being used from the given power profile. The app corresponding to output class label is considered as the app being used at the particular time.

**Performance metric**

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.

**Experimental results**

**1) Impact of window length**: 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. A smaller W means that the attacker only needs to collect a power profile for a shorter period, making the attack more practical.

**2) Impact of sampling frequency**: The higher fs, the finer-grained characteristics of the collected power profiles. A higher fs leads to more power profiles that need be stealthily transmitted over the Internet, making our attack easier to detect.

**3) Impact of number of training instances**: We can see that the average identification rate increases as N0 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.

**4) Feature importance**: We have also studied the importance of different features used in POWERFUL, characterized by information gain. The higher information gain, the more important a feature is, and vice versa. The less important features (i.e., with small information gain) still affect the overall classification results.

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

**APP fingerprinting**

Our work is also related to the line of research in app fingerprinting, which aims at identifying apps through traffic analysis.

**Power analysis**

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.

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 novel attack that reveals user locations via power analysis of the user’s smartphone. Assuming that the distance between and 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.

2. Profiling Power Consumption on Mobile Devices ENERGY 2013

The main contributions of this paper are:

* A profiling test bench, which allows executing defined scenarios on mobile devices and profiling the related power consumption through external measurement hardware;
* A comparison between power consumption of two different generations of Android OS-based mobile devices validated by statistical analysis of gathered data.

Usually, an accurate power consumption analysis of mobile or embedded devices is component-based. However, instantaneous information about discharge current and remaining battery capacity is not always available, because most devices do not have built-in sensors to collect these data.

Using external metering instrumentation to detect power consumption.

Profile power consumption of applications, basing upon their component usage.

We selected two independent variables: the smartphone model (M) and the specific scenario (S). Each has been executed 30 times, with a fixed duration of 4 minutes per scenario.

S0: Standby. This scenario provides the baseline for our analysis. During this scenario, there are no user applications in execution, and 2G and 3G connections are enabled.

S3: File download through Wi-Fi connection. In this scenario, the scheduled task launches a new thread, which downloads a remote file, the Ubuntu 11.10 disk image, up until the scheduled timeout (4 minutes).

S7: Scan for Bluetooth devices. In this scenario, a scan for Bluetooth devices is performed. The scan process lasts, according to specifications, 12 seconds in average. At the end of the scan procedure, the task simply restart, up until the prefixed duration.

S8: CPU-intensive activity. The aim of this scenario is maintaining a high CPU workload while gathering power consumption data. For this purpose, repeated cryptography operations are performed, with a pool of 20 threads, each of them iterating the procedure 10 times.

S10: Active display with 50% Brightness. The aim of this scenario is assessing the impact of the active display over power consumption. This scenario is similar to S0, the only difference being that all radios (2G, 3G, Wi-Fi) and the SIM card were disabled.

**Hardware Instrumentation**

The power consumption data was acquired through a power metering architecture. The battery was removed from the devices, in order to avoid bias due to discharge and subsequent OS power saving procedures. The battery terminals were directly connected to a DC power supply, providing 5V steadily.

The DC power supply used is the TPS-2000D produced by Topward Electric Instruments Co. A Data Acquisition Board (DAQ), the DAQ Lite produced by Eagle Technology, was used to acquire the power consumption data.

The DAQ was set to a sampling frequency of 350 HZ, in order to produce an amount of data statistically relevant, but not prohibitive for subsequent computation.

**Software Setup**

In order to automate scenario execution in our experiments, a supporting software environment was developed, composed of two Android applications, a server side applications and macro scripts, to be executed by the tool AutoHotKey 3. The developed Android application allows enabling or disabling components, such as Bluetooth, GPS or Wi-Fi interface, in order to avoid bias during scenarios that do not use them.

These applications communicate with a server machine, which is then connected to the DAQ via USB. the server application then launches a AutoHotKey script that performs the needed operations for data acquisition and logging.

**Instrumentation and Experiment Design**

The goal of data analysis is to apply appropriate statistical tests to reject the null hypothesis. As we expected, the collected power consumption values, for both smartphones, do not follow normal distribution. This was verified by means of the Shapiro-Wilk test, with a resulting p-value lower than 0.05.

**Threats to validity**

We will classify threats of experiment validity in two categories: internal threats, derived from our treatments and instrumentation, and external threats that regard the generalization of our work.

A possible internal threat concerns the sampling frequency adopted by the DAQ, namely 350 Hz. We chose this frequency value for practical reasons, in order not to obtain a huge amount of data, which could not be computed in a reasonable time by our servers.

However, this frequency, compared to the operational frequencies of the selected smartphones, could be seen as quite low. A more significant threat comes from the usage, in some of our scenarios, of different communication networks, which are characterized by an unpredictable behavior.