Pavel Lifshits *, Roni Forte , Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein

Power to peep-all: Inference Attacks by Malicious Batteries on Mobile Devices

Abstract: Mobile devices are equipped with increasingly *smart batteries* designed to provide responsiveness and extended lifetime. However, such smart batteries may present a threat to users' privacy. We demonstrate that the phone's power trace sampled from the battery at 1KHz holds enough information to recover a variety of sensitive information.

We show techniques to infer characters typed on a touchscreen; to accurately recover browsing history in an open-world setup; and to reliably detect incoming calls, and the photo shots including their lighting conditions. Combined with a novel exfiltration technique that establishes a covert channel from the battery to a remote server via a web browser, these attacks turn the malicious battery into a stealthy surveillance device.

We deconstruct the attack by analyzing its robustness to sampling rate and execution conditions. To find mitigations we identify the sources of the information leakage exploited by the attack. We discover that the GPU or DRAM power traces alone are sufficient to distinguish between different websites. However, the CPU and power-hungry peripherals such as a touchscreen are the primary sources of fine-grain information leakage. We consider and evaluate possible mitigation mechanisms, highlighting the challenges to defend against the attacks.

In summary, our work shows the feasibility of the malicious battery and motivates further research into system and application-level defenses to fully mitigate this emerging threat.

Keywords: Malicious battery, Power side-channel

DOI 10.1515/popets-2018-0036

Received 2018-02-28; revised 2018-06-15; accepted 2018-06-16.

***Corresponding Author: Pavel Lifshits :** Technion
Roni Forte : Technion
Yedid Hoshen: Hebrew University
Matt Halpern: UT Austin
Manuel Philipose: UT Austin
Mohit Tiwari: UT Austin
Mark Silberstein: Technion

1 Introduction

We study a new attack vector to launch power side channel attacks on mobile devices – a smart battery that includes storage and processing elements to stealthily monitor and report user activity, in addition to their benign power management functions. As smart batteries with increasingly sophisticated logic are gaining popularity [24, 25], the attack we study may turn such batteries into a surveillance device. An attacker may slip a malicious battery into the phone with only brief physical access, e.g., an interdiction attack [15] in the supply chain or at an airport security check. The battery may then monitor the user's activity and use covert channels to communicate the collected private information to the remote attacker.

A malicious battery may serve as a powerful platform for attacks on user's privacy. First, such an attack is hard to detect because it leaves no software footprints on the device. Second, unlike other attacks that require attaching probes to power charging cable [37] or intercepting network traffic [35], the malicious battery may continuously monitor the phone's activity. Third, the attack does not involve intrusive hardware modifications to the phone other than replacing its battery. Unlike other rogue hardware attacks [27], replacing the battery is usually a simple procedure that requires no special equipment. Finally, all the phone's activities are exposed, therefore the attacker may amplify the power of each individual attack by combining several inference attacks together. For example, the attacker may combine the keystroke inference and the website inference to dramatically improve their precision.

The attacker, however, faces several unique challenges. First, the malicious power trace acquisition and processing device must fit the small form factor and power envelope of the phone's battery package. This constraint limits the signal sampling rate and the computational complexity of the classification algorithms. We find that the attack can be carried out while sampling the power at a sampling rate two orders of magnitude lower than reported before [37].

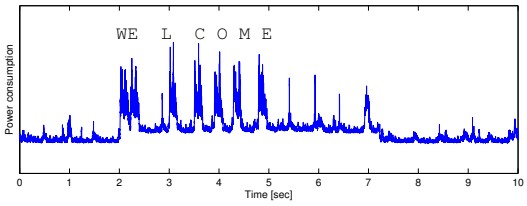

Fig. 1. Power trace of typing the word "welcome"

Second, the attacker continuously monitors the phone's power draw; therefore she must localize the actual events and filter out all the *irrelevant* ones while processing only those of interest. Since the power signal is noisy, this *open-world* classification problem is significantly more challenging than the *closed-world* problem of distinguishing between different websites in prior works [37].

Finally, data exfiltration from the battery poses a challenge. One option is to add an out-of-band transmitter to the battery (e.g, Bluetooth or WiFi). However it might reveal the presence of the malicious battery. Another alternative is to manipulate the output voltage and current from the battery to covertly communicate with an app installed on the phone via power monitoring software interface. However, the app needs to be installed on the phone, which falls outside our threat model that precludes software installation.

This paper addresses these and other challenges, and demonstrates that the malicious battery attack enables the attacker to build a detailed profile of the phone's user. We describe a general data processing framework for the attack, implement several novel inference attacks, analyze the sources of the information leakage on the phone, and suggest possible defenses.

We make the following contributions:

Attack vector. We propose a new stealthy attack vector to launch power channel attacks on mobile phones – a malicious battery that an attacker can insert in place of an existing battery. This attack is external to the phone and thus undetectable by the software tools, and can be built using cheap off-the-shelf components.

Recovery of diverse information. The attack recovers a variety of private information. We show several novel examples: (1) soft keyboard keystroke inference (Figure 1) that recovers up to 36% of the typed characters, including word delimiters with 88% accuracy, and reduces the password search space by three orders of magnitude on average; (2) website inference from a watchlist of Alexa top 100 popular websites in an open-world scenario with 65% precision on average, (3) incoming phone call and camera shot detection, including the fine-grain lighting conditions that enable distinguishing between indoor and outdoor shots.

End-to-end attacks. We develop a general data processing pipeline for power side channel inference attacks. The framework includes three stages: activity detector, novelty detector and a classifier. These stages are implemented for each attack. We implement them for two end-to-end attacks, web inference and keystroke inference, and evaluate them on 1-hour raw power traces of a regular phone activity. We demonstrate successful attacks on three popular mobile phone models by Samsung and Huawei and show that they are robust to the sampling rate reduction down to 100Hz.

Battery-browser-website exfiltration channel. We implement a novel covert channel from the battery via a web browser to a malicious website. To circumvent the browser security limits on the battery level sampling rate, we build a circuit to directly manipulate the *battery charging state* from inside the battery, by exploiting the phone's wireless charger circuitry. This allows sending up to 1 bit every two seconds, 15× the bandwidth achievable via the battery charge level JavaScript Battery Status API. The covert channel is bidirectional; thus, the exfiltration mechanism can be triggered when a visit to a malicious website is detected by the battery.

Sources of information leakage. We analyze the architectural sources of the attacks. Specifically, we measure the power draw by individual SoC components using a development board. We find that the CPU is best correlated with the overall power draw, but GPU and DRAM power traces alone are sufficient to infer webpage identities with almost 75% precision. The keystroke attack is possible because of the touch screen power draw during key presses, and does not depend on the CPU.

Defense. We show that using randomized Dynamic Voltage and Frequency Scaling (DVFS) reduces the accuracy of the website inference. However, it is not effective against attacks such as keystroke inference that do not rely on the CPU power consumption. A more targeted keystroke-specific defense we implement does protect from the keystroke attack. These are partial solutions, but they highlight the challenge of strong protection in the face of increasingly powerful machine learning and signal processing techniques.

This work shows that the emerging battery technology may enable a new type of power side channel attacks on mobile devices, and motivates further research to alleviate this threat.

2 Motivation

Threat model: interdiction attacks. An attacker with brief physical access to the mobile device – at the supply chain, repair shops, workplaces, etc. – can replace the device's battery. This is an example of an *interdiction attack*. Interdiction attacks using malicious VGA cables have been used to snoop on targeted users [15]. Malicious battery introduces a new attack vector into a mobile device.

This hardware-based attack is stealthy – it requires no software components to execute on the phone, has small hardware footprint, and the malicious hardware itself consumes only a few milliwatt (§12), hardly noticeable by the victim. Moreover, the attack targets the phone part often produced by third-party vendors, making it hard to detect even by close inspection.

The attack's low cost and its reliance on replaceable batteries make it affordable for small-scale attackers. For example, an attacker might sell the batteries on-line, attracting clients by lower cost or extended warranty.

One of the most appealing aspects of the battery-based attack is that the power side channel it exploits is all-embracing: all the phone activity is exposed. Thus, the attacker may correlate the information obtained from multiple attack vectors. For example, she can identify the context of the keystroke (e.g., the website being visited), and the events that preceded or followed it in time, such as a camera shot or a phone call. Together, these pieces of information reconstruct a coherent portrait of the user's activity, dramatically amplifying the power of individual attacks.

Fig. 2. Microcontroller inside the battery of Samsung Galaxy S4

Hardware requirements. Embedding modest computing capabilities in a battery is already feasible (see Figure 2). Moreover, *software-defined* batteries [1] leverage more advanced microcontrollers to squeeze efficiency from a heterogeneous set of batteries.

The form-factor and power constraints put limits on the sampling rate of the phone's power draw. This rate dictates the processing capacity, power draw, and storage requirements of the malicious circuit. All the attacks in the paper are performed at 1KHz, and they are quite robust to the sample rate reduction (§8). At this rate, no expensive or bulky sampling hardware is needed. Further, the trace can be processed in real time or recorded over long periods. See § 12 for detailed analysis.

Exfiltration scenarios. For a targeted attack, an attacker may physically access the phone to fetch the malicious battery with the recorded power trace. Alternatively, the battery may manipulate the reported charge level to force the phone's owner to connect to a malicious charger, which in turn may relay the data to an outside attacker [30]. To trigger this behavior in a specific location equipped with malicious chargers, the battery may identify the browsing event to a gateway for connecting to a public hotspot.

For a general attack, the exfiltration requires from the attacker to communicate with the battery remotely. The battery may establish a covert channel to an app by encoding information in the charge level exposed to the phone's software. However, our threat model assumes no malicious software installed on the phone.

Instead, the attacker may embed a malicious JavaScript on her website and use the Battery Status API to retrieve the battery state, implementing the convert channel via the browser. The phone's battery detects that the victim visits the website with the malicious JavaScript (e.g., a public hotspot gateway), and transmits the data. We consider this scenario in §9.

Defense. Power normalization is a common technique to ensure privacy in cryptographic devices, but a general purpose computing system like a smartphone cannot be run at peak power just for privacy. One option that we evaluate is to obfuscate the power trace by randomizing the dynamic voltage and frequency scaling settings. Unfortunately, it is not effective against all the attacks, which require more targeted defense (§ 11).

3 Related Work

There is a large body of research on mobile phone inference attacks in general, and power side channel attacks in particular (see Spreitzer et al. [31] for a comprehensive survey). We briefly discuss the prior art relevant to this paper and highlight the main differences.

Power side channel attacks on mobile devices. A number of power side channel attacks have been published. However, the attacks' targets, power acquisition methods, setups, exfiltration methods and processing tools differ from ours. Powerspy [18] infers the user's driving route by sampling the power via the Android

power monitoring API. Clark et al. [6] uses AC power consumption to identify 50 webpages with 87% precision. The threat model is based on a 250kHz sampling rate (4Mbps), and a complex sampling device to launch the attack. Our work differs in several respects. We use DC power sampled at 1KHz and achieve comparable precision (up to 77% for 100 pages). As a result, our attack can run on simple hardware that satisfies the energy/form-factor constraints of a malicious battery. We also show other inference attacks, analyze the sources of the information leakage, and suggest defenses.

Yang et al. [37] and Chen et al. [5] use power trace to infer browsed websites and active apps respectively. While their goals are similar to ours, their methods are inapplicable to malicious battery settings because they focus on a *closed-world* setup. Thus, they can only *distinguish* between known activities from the dictionary, while our attack handles arbitrary inputs. Furthermore, their classifiers would not be effective in the open-world scenario for noisy signals such as power.

Sensors. Motion sensors and accelerometers have been used for numerous attacks, for example, to infer keystrokes [19, 36], passwords [23], and approximate location and driving trajectory [14]. These attacks require an active malicious application/web page to sample sensor's values. In contrast, a malicious battery continuously monitors the activity in the background, and connects to a specific website only for exfiltration.

Keystroke inference. Keystroke timing attacks infer typed characters from the timings of keystrokes. Song et al. [29] has pioneered this approach to predict key sequences from the inter-keystroke timing of an SSH session. Several followup works focus on the ways to obtain the keystroke timings to enable similar attacks. Specifically, Foo et al. [10] use the interrupt-timing side channel, and Zhang et al. [38] leverage the process stack information available in `/procfs` in multi-user servers. Similar attacks have been demonstrated on mobile devices with comparable accuracy. Diao et al. [7] present inference attacks on Android through interrupt timing analysis, and Simon et al. [28] utilize interrupt timing to infer text entered through gesture typing on Android virtual keyboards. To the best of our knowledge, ours is the first attack that infers keystrokes from the power trace. Moreover, our attack does not explicitly rely on timing between the strokes, but uses a convolutional neural network to infer keystrokes directly from the power signal, which reduces the training complexity by allowing training from typed text.

Attacks on cryptographic hardware/software. Power side-channel attacks have targeted fixed-function (cryptographic) hardware and embedded devices [3, 20]. However, most attacks and defenses are not directly applicable to a general-purpose program on a mobile device. Another attack vector is electromagnetic emanations, such as the one by Genkin et al. [12] that demonstrates ECDSA key extraction. These attacks specifically target cryptographic software primitives.

Defenses. Most defense techniques are for attacks on cryptographic applications, and are less applicable here. For example, one option is to modify the software to use constant execution path code, or use instructions with a less leaky power profile. This is not applicable to closed-source general-purpose applications. Alternatively, instead of reducing the signal, a defender can increase the (ideally, random) noise in the power trace [34]. Our defense follows the same principles.

4 Attack Overview

The general scheme of the attack is depicted in Figure 3.

Acquisition. An attacker builds a training set by collecting the power trace while performing the target activity (e.g., web browsing or typing).

Offline training. For each type of activity an *attack-specific* classification pipeline is trained. Each attack-specific pipeline must be able to function in an open-world setup.

Inference. The attacker continuously samples the battery draw on the victim's device. The trace is processed by all the classification pipelines of all the attacks independently. Each pipeline watches for the user activity relevant to it, using the respective *activity and novelty detector*. Once the activity is detected, the pipeline processes the events. We note that the events inferred in one attack-specific pipeline may improve the accuracy of another pipeline, e.g., keystrokes on a particular webpage may be easier to guess.

4.1 Classification pipeline overview

All the attacks are built using the same three-stage processing pipeline, where each preceding stage is trained to narrow down the set of traces for processing in the following stage.

The activity detector continuously monitors the power draw at the granularity of few milliseconds (e.g, 100 ms for the webpage inference pipeline). Once the relevant activity has been detected, the system processes

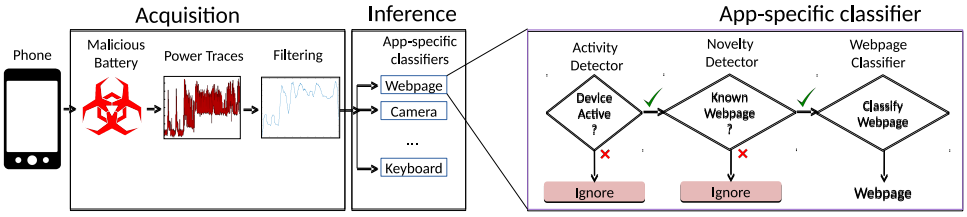

Fig. 3. Attack overview

the trace via a sliding window for a predefined amount of time. For example, we find that 15 seconds is enough to fully load most webpages. Thus, the webpage classification pipeline records the window of 15 seconds of the trace every time the web browsing event is detected.

The novelty detector distinguishes the traces that correspond to some unrelated phone activity from those that correspond to the events potentially relevant to the attack's pipeline.

Finally, *the classifier* labels the traces by inferring the events of interest.

In this paper we show several pipelines. We explain in detail two end-to-end pipelines for the web site inference (§6.1), and the keystroke inference (§6.2) and then show and evaluate only the classification stages of the photo shot and phone call attacks (§6.3).

5 Experimental setup

We evaluate the attacks on three popular mobile phone models [17] running different versions of the Android OS and featuring different hardware specs (see Table 1).

Phone Configuration. We use the phone's default software stack and default OS settings, including interactive DVFS power governor and enabled background services, in particular, the Gmail app configured with an active user account. The phone uses wifi to connect to the Internet.

Recording power traces. We build a simple acquisition device comprised of a shunt resister, signal conditioning stage (amplification and filtering), and an ADC. The device is attached between the positive battery terminal and the phone. See §12 for technical details. Unless stated otherwise, we sample the output using NI myDAQ at *1KHz* with 16 bits per sample.

Battery vs. controlled power supply. For convenience we connect the phones to a controlled power supply that replaces the battery. This setup eliminates the power drift of the real battery-only power supply, which might seem favorable for the attack. However, we evaluate the attacks in both setups and observe no statistically significant difference in the results.

Mobile Device	Huawei Mate 9	Smsng Galaxy Note 4	Smsng Galaxy S4
Chipset	Hisil. Kirin 960	Snapdrgn 805	Snapdrgn 600
CPU	Cortex-A73 & Cortex-A53	Cortex-A57 & Cortex-A53	Krait 300
Display	5.9'	5.7'	5.0'
OS	7.0 (Nougat)	5.1.1 (Lollipop)	4.4.2 (KitKat)
Browser	Chrome 53	Native 6.2 / Chrome 63	Native 2.1 / Chrome 43
Battery	Li-Po 4000mAh	Li-Ion 3220mAh	Li-Ion 2600mAh

Table 1. Hardware and software used for evaluation

Online vs. offline attack. The attack can be performed offline or online. In the offline case, the power trace is recorded on the battery, and the attacker physically retrieves the battery for offline processing. In the online case, the trace is processed on the battery and the results exfiltrated via the covert channel (§9).

Evaluation methodology. All the experiments are performed offline. For each, we execute the desired phone activity either manually (e.g., text typing) or by sending the respective Android intents (e.g., to browse a website) from a remote machine, while logging the "ground truth" label for each event. In parallel we record an unmodified power trace by sampling the power consumption of the phone from its battery and storing the samples in a file. To evaluate the attack we replay the entire trace into the respective processing pipeline, and compare the output labels with the respective ground truth labels.

6 End-to-end evaluation

In this section we overview the main results. Section 7 elaborates on the technical details and Section 8 discusses the robustness of the attacks.

% of known websites in the test set	0%	25%	50%	75%	100%
Precision [%]	77%	71%	65%	60%	53%

Table 2. Precision of the end-to-end attack with a different proportion of webpages from the watchlist.

6.1 Website inference

We evaluate the accuracy of the end-to-end attack for detecting the Alexa top 100 popular websites (their landing pages) in an open-world scenario. We create an hour-long activity session with three types of events: browsing webpages in the watchlist (Alexa top 100), browsing webpages not in the watchlist (Alexa top 101-200), and invocations of two popular games – Temple Run and Angry Birds. The websites in the watchlist are called *known* and should be correctly labeled, whereas all the other events should be marked *unknown*. The time between the events is normally distributed $N(30s, 15s)$.

Software configuration. We use the default configuration with the browser cache enabled. We automate the experiment by running a shell script on the phone that initiates the browsing by sending the "android.intent.action.VIEW" intent. To emulate an interactive browsing session we keep the phone's screen permanently turned on using the StayAlive app.

Accuracy calculation. We compute the attack accuracy for each test trace by counting the number of correctly labeled webpage visits. The label is considered correct when: (1) a known webpage is labeled with the correct name, (2) an unknown webpage is either ignored or labeled unknown (3) any other phone activity is either ignored or labeled as unknown. Conversely, we consider a misclassification to be any incorrectly labeled browsing event, any undetected browsing event, or any nonexistent event labeled as one of the webpages.

6.1.1 Results

We record 1-hour long power traces of several sessions while varying the proportion of known and unknown events, and feed them into the processing pipeline (§7.1). Table 2 shows the precision of the end-to-end attack on Samsung Galaxy Note 4. The results on the other phones are within ±3%. On average, the attack correctly labels 65% of the browsing events across different test sets. The fewer known websites there are in the test set, the higher the precision. This is because the unknown websites are filtered by the novelty detector alone. As a result, only the activity and novelty detectors are active, allowing for higher end-to-end precision. The traces which contain known events, however, exercise all the three pipeline stages combined, and, consequently, accumulate the error from all of the stages.

Our notion of precision matches the intuition: 65% precision end-to-end means that 65 out of 100 activities in a one-hour trace were labeled correctly. However, this is a rather conservative coarse-grain measure. First, it hides the fact that some webpages are better classified than others. In fact, some webpages are labeled correctly in 100% of cases, while others are misclassifed.

We also normalize the correctly classified events by the number of actual browsing events. In practice, however, the pipeline evaluates 36,000 fragments in one hour (every 100 ms), filtering out all but about 100 fragments that correspond to real events. If we count all those evaluated fragments as correctly labeled, the pipeline accuracy approaches 100%. Although this value does not reflect the actual accuracy of the attack, it does indicate that the browsing activity can be distinguished with very high precision.

Finally, the actual classification task performed by the pipeline is quite complex. For each fragment in the power trace, one of the 101 labels (100 watchlist webpages and "unknown" for all other events) is assigned. Hence, a success rate of 65% is 65 times higher than the 1% probability to obtain the correct label at random.

6.2 Keystroke inference

We train the classifiers on the labeled power trace (key press, key release and key character) of a specific user typing English text. The text is built such that each character appears at least 25 times. The set consist of 1200 words and 6964 characters.

Soft keyboards. The reported results are for a Huawei Mate 9 phone using the default Gboard soft keyboard. Similar results are obtained with SwiftKey soft keyboard on Huawei Mate 9, and the Samsung default keyboard on Samsung Galaxy Note 4.

6.2.1 Results

We perform three experiments: (1) detection of characters in a free text, (2) password cracking, and (3) inference with a constrained dictionary.

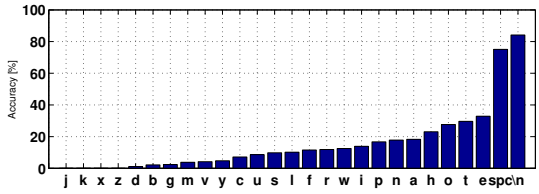

Fig. 4. The accuracy of the keystroke inference per character.

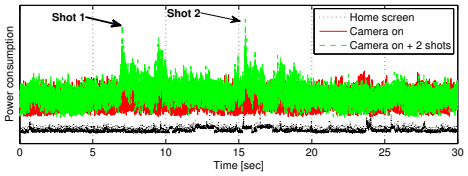

Fig. 5. Example of a power trace during a camera shot.

Free text inference. We record a 30 min-long raw power trace created while typing English text with 2400 characters and feed it into the classifier. The attack correctly guesses 36% characters, which is an order of magnitude better than a random guess. Figure 4 shows the prediction accuracy for individual characters. "Space" and "Enter" result in the best precision: 75% and 84%, respectively. This experiment shows that the attack is able to recover some information about the text, but cannot recover the original text, even after spell-check.

Password search space reduction. We randomly generate 50 character sequences with a uniform distribution of characters 10-13 letters long. We deliberately choose less popular passwords for this experiment because guessing more popular ones might be easier using other methods.

We calculate the search space reduction as follows. The attack outputs a list of the candidate characters for each input character. The candidate list is sorted according to the classifier confidence value from the most likely to the least likely candidate. Let c_i be the location of the true letter in the candidate list for the character i in the input word. The number of more likely candidates that precede c_i is denoted l_i. We assume that these candidate lists guide the exhaustive search when cracking the password. Thus, the search encounters the correct password only after scanning through all the candidates for the largest $L = max(l_i)$ among all input characters. Therefore, we compute the search space reduction as $(\frac{28^{|w|}}{L^{|w|}})$, where $|w|$ is the word length. We note that the attack reliably estimates $|w|$, which alone improves password guessability. However we do not consider this factor in the search space reduction metric.

The geometric mean of the search space reduction is $O(10^3)$, with up to $O(10^4)$ reduction in the best case, and $O(10^2)$ in the worst case. These results are comparable with the results of other key timing inference attacks. Specifically, Song et al. [29] report 50× search space reduction on 7-8 character length passwords by inferring key timing from the SSH network transactions. Zhang et al. [38] infer the key timing from `/procfs` and report $200\times - 2000\times$ search space reduction on 15-character passwords.

Closed dictionary. We evaluate a scenario where a user types words from a known dictionary, for example, a flight destination on a flight booking website, or a drug name on in a drug description portal [1]. Thus, inferring the typed data may reveal sensitive information. In this scenario, the website inference attack can be used to narrow down the context of the typed sequence, helping the attacker identify the closed dictionary used by the victim.

We evaluate this attack using the list of all 154 destinations (1606 characters) from `www.EasyJet.com`. We type them all and infer them from the power trace. The experiment involves two stages: guessing the keystrokes via the power trace, and then finding the closest neighbor (Levenshtein distance) from the dictionary.

The attack correctly recovers 18% in the Top 1 guess, 30% in the Top 2, 40% in the Top 3, and 50% in the Top 5. These results are about a 20-fold improvement over random guess.

6.3 Other attacks

We briefly outline two other attacks that demonstrate the diversity of the information that can be recovered from the power channel.

6.3.1 Camera

We evaluate the attack that determines (a) that a shot was taken (b) whether it used flash (c) its lighting conditions.

The power footprint of the shot is shown in Figure 5. It is clearly visible when the camera is active (green line) and when the shots are taken (at 7.5 sec and at 15 sec).

Flash/no flash. We take 60 shots of the same object, half with and half without using flash. Unsurprisingly,

1 https://druginfo.nlm.nih.gov/drugportal

the flash power footprint is rather unique, so the flash state is classified correctly in all the test cases.

Precision	Recall Sensitivity TPR	Accuracy	Specificity
78%	90%	86%	74%

Table 3. The camera attack: distinguishing lighting conditions.

Identifying lighting conditions. To discover which properties of a shot can be determined, we perform a controlled experiment in which we take 50 shots of the same scene under different lighting conditions: (1) during daylight with fluorescent lights, and (2) with window blinds closed and lights turned off. Table 3 shows that the lighting conditions can be inferred with high precision.

Determining lighting conditions can be useful, for example, to identify indoor-vs-outdoor shots taken during the day. To show that, we take 200 pictures, 100 indoors and 100 outdoors. The indoor pictures are taken under normal lighting conditions, in an office and in a large open space, and contain people as well as various objects such as shelves, books, walls. The outdoor pictures contain trees, buildings, and people. Here 91% of the outdoor shots, and 100% of the indoor shots are identified correctly.

We conjecture that lighting conditions are the main source of the signal. We find that minor changes in the scene or the distance to the target do not show statistically significant changes in the power signature.

6.3.2 Incoming phone calls

This attack recovers the communication profile by identifying incoming calls. We train our classifier on a labeled power trace of 24 incoming calls. There are four groups of settings (6 calls per group): two different ringtones, sound muted, and with vibration alone. We then evaluate it on a 10 minute trace during which 15 incoming calls are received. In the evaluation we use a ringtone that is different from those two used in the training, and also test both mute and vibration only traces. The attack correctly identifies the incoming calls in 100% of the cases.

Figure 6 explains the reason for such a high precision. It shows t-SNE visualization of the low-dimensional embedding of the power traces with incoming calls, compared to the traces when the phone is idle and while typing. The traces of the incoming calls are clustered.

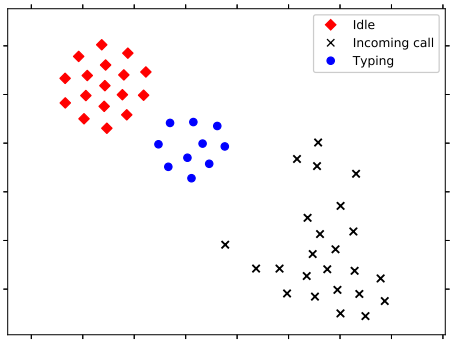

Fig. 6. t-SNE visualization of the low-dimensional embedding of the power traces of incoming calls vs. phone in idle mode vs. user typing. The traces of the same activity are naturally clustered.

7 Technical details

We describe the signal processing techniques and provide more detailed analysis of the results. Throughout this section, when presented with the choice of signal processing and classification techniques, we deliberately choose those with lower computational complexity to enable the execution of the attack using the device with low power consumption and compact form-factor to fit in the battery, as discussed in §12.

Signal representation. There are many known representations that reduce the dimensionality of time series while preserving the fundamental characteristics of the data, as well as many possible similarity distance measures [8]. We consider several options for representing and matching power traces: (1) spectral based representations (such as DFT), (2) various distance measures (such as DTW [2]), and (3) Singular Spectrum Analysis (SSA) as used by Genkin et. al [12]. However, these techniques result in insignificant (1-2%) improvement in the accuracy while being computationally demanding. Therefore we use a simple average-based denoising approach and Euclidean distance.

7.1 Website inference

We explain the attack-specific pipelines in detail, and evaluate each stage in isolation.

7.1.1 Activity detector

We define the phone activity as any action performed by the user, e.g., web browsing, running an application, or reading an email. When the activity is detected, the detector feeds the adjacent 15-second fragment to the next processing stage, dropping all the rest.

Implementation. The detector analyzes the power trace every 100 milliseconds using a sliding window with 15 ± 1 seconds of samples. The detection comprises the following steps.

1. **Smooth signal** by computing a running average with 1000 samples.
2. **Find peaks** above a certain threshold, set according to previous traces of browsing events.
3. **Test energy** of the signal in the window of 15000 samples, every 100 samples in the smoothed trace, starting 1000 samples from the peak. The earliest point when the signal energy exceeds the threshold is labeled as the start of the browsing event.

Evaluation. We consider the detection of the browsing event within 300 milliseconds of the ground truth as correct, because the actual signal used for the next stages is smoothed using a running average window of 300 samples as we explain below. We evaluate the activity detector on *15 hours* of the raw power traces used in the experiments (540,000 overlapping 15-second fragments) and achieve 100% precision and recall.

7.1.2 Novelty detector

The novelty detector receives the original (non-smoothed) 15-second power trace fragments and acts as a binary classifier between known and unknown events.

Watchlist dictionary. We collect the power traces while browsing the websites in the Alexa top 100 websites. The watchlist dictionary contains the labeled power traces of 10 independent visits to each website (total of 1000 browsing events for 100 websites), the first 15K samples. Some websites, e.g., YouTube, prompt to install the relevant app on the first visit. We discard the prompt to redirect all the subsequent visits to the mobile version of the page.

Implementation. The novelty detector is implemented as a set of 100 1-versus-all Support Vector Machine (SVM) binary classifiers with Radial Basis Function (RBF) kernels. Each classifier i is trained for each webpage W_i in the watchlist (hence 100 classifiers), so the i_{th} classifier's output is either "not W_i" or "W_i". For each input all the classifiers are evaluated at once, and the input is labeled "unknown" only if classified as "not W_i" for all i. Other options, such as majority vote among classifiers, perform worse.

Actual \predicted	**Unknown**	**Known**
Unknown	77% ± 8%	29% ± 1%
Known	23% ± 8%	71% ± 1%

Table 4. Novelty detector performance.

Training. The classifiers are trained separately. We assign weights to the positive samples of the respective webpage. The weights are inversely proportional to the ratio of the positive samples in the complete set. For example, our dictionary contains 10 positive examples for each webpage, and 1000 traces overall. Thus, the weights we assign to the traces of webpage W_i when training the classifier i are 990/10. The use of weights improves the results significantly, compensating for the small number of positive samples in the training set.

Evaluation. Our evaluation is the standard 10-fold cross-validation performed 10 times for different selection of unknown sites. Specifically, we first select 100 traces of 10 randomly selected *webpages* (not traces) which we use as true "unknown" and remove all their traces from the training set in this iteration. In addition, we randomly select 10% of the remaining 900 *traces* as true "known" traces, and also remove them from the training set. The final result is the training set with 810 traces, and the test set with 90 samples of "known" and 100 samples of "unknown" webpages. Known webpages have a few of their original 10 power traces in the training set, whereas unknown webpages have none. We then train 90 classifiers corresponding to all the "known" webpages, and evaluate the novelty detector. We repeat this procedure 100 times – for each new choice of the known and unknown traces in the test set – and compute the average and standard deviation across all the performance metrics.

Table 4 summarizes the results. The novelty detector success rate is about 72%, with slightly better classification of visits to unknown webpages. This is not surprising because the training set is significantly biased toward unknown samples. On the other hand, such a novelty detector performs better in the actual attack in which the vast majority of the activities are truly unknown.

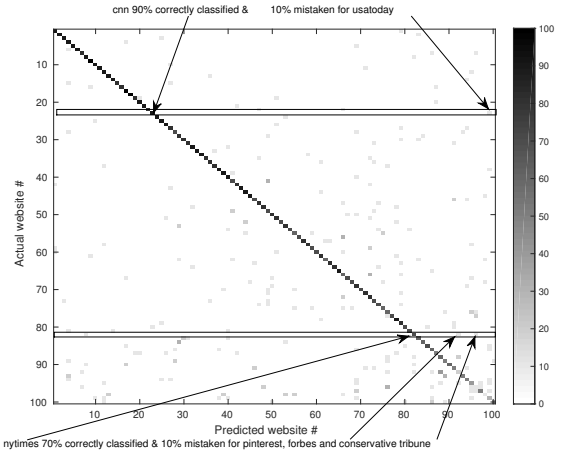

Fig. 7. Confusion matrix for the classification stage.

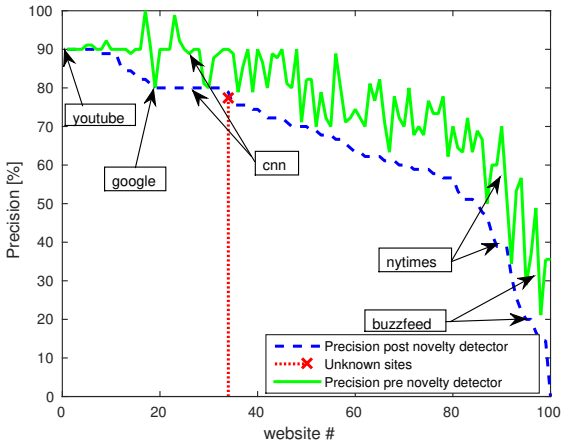

Fig. 8. Novelty detector and classifier combined

7.1.3 Classifier

The classifier processes the traces which the novelty detector labels as "known". The classifier uses a simple 1-Nearest Neighbors algorithm, trained to recognize the webpages in the watchlist dictionary.

Evaluation. We evaluate the classifier using the standard 10-fold cross-validation. We present the confusion matrix in Figure 7, and the precision per webpage in Figure 8. For completeness we also show the precision of the novelty detector on the same graph (marked with the cross sign). Unsurprisingly, certain webpages are classified significantly better than the others. For example, youtube.com is classified correctly in 90% of the visits, while buzzfeed.com is classified correctly only in 26%.

Analysis. We conjecture that webpages are distinguishable mainly by their images and JavaScript components. To confirm this conjecture, we record the power traces while browsing with image loading and/or JavaScript execution disabled in the browser. To reduce the effects of network latency variations, we install a caching proxy on a local server. We browse through the caching proxy after all the requested pages have been prefetched.

We find that disabling both JavaScript and image loading reduces the classification accuracy from 77% to 21%, whereas disabling each one separately results in the accuracy of 44% and 52% respectively. This indicates that these two features together are indeed the dominating factors contributing to the unique webpage power signature. Thus, we believe that the low classification precision for certain websites is partly due to their frequent content changes which occurred over the course of the experiment.

7.1.4 Classifier and Novelty detector combined

Figure 8 shows the per-webpage classification success rate of the novelty detector and classifier combined (denoted post-novelty detector in the graph). As expected, the addition of the novelty detector reduces the overall precision. More interestingly, the performance degradation is inconsistent across the webpages, since the novelty detector may incorrectly label a webpage as unknown, even though the classifier in isolation is likely to classify it correctly.

7.2 Keystroke inference

The keystroke pipeline comprises two stages: keystroke detector and keystroke labeler. The former identifies the fragments of the power trace that contain a keystroke. The labeling classifier is used to label the fragments with either English character, "Space" or "Enter".

7.2.1 Keystroke detector

The detector is a binary classifier that receives a power trace fragment of 100 samples (100 ms) and labels the fragment as a "keystroke" or "not-keystroke". We use the 100 ms window because we find that this is the shortest time that the key remains depressed (i.e., the time between the press and release events).

We use a Convolutional Neural Network (CNN) with a typical topology [16]: five convolutional layers followed by two fully connected layers and a softmwax layer with two outputs. Each convolutional layer has 128 1x7 filters followed by 50% pooling, ReLU activation, batch normalization, and 20% dropout.

The network is trained end-to-end using the cross-entropy loss function. The ground truth input fragment is tagged according to the activity (key depressed/released) in the center of the fragment.

The keystroke detector finds continuous power trace fragments that enclose a complete keystroke (i.e., all the samples when the key is depressed). Since the keystroke lasts on average 100± 15ms, we search for a continuous fragment of at least 80 samples such that at least 98% of them are marked as keystrokes. These fragments are then fed into the keystroke labeler stage.

We consider the fragment classification to be successful if the fragment indeed encloses the keystroke. When testing on a text with a total of 2664 characters the classifier achieves high precision of 99.99%.

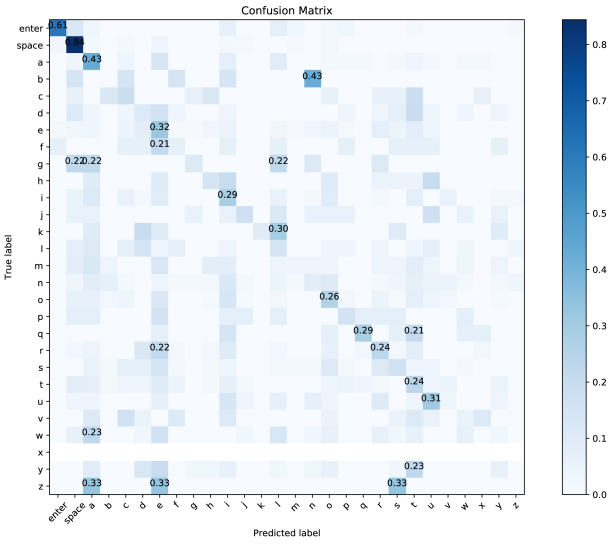

Fig. 9. Confusion matrix for the keystroke detection for English text with 1000 characters, including Space and Enter. Only the probabilities >0.2 are shown. The ideal classifier would have 1-s (darkest color) only along the diagonal.

7.2.2 Keystroke labeler

The labeler uses another CNN with the same architecture, only with much larger input of 2,000 samples and 8 convolutional layers. The convolutional layers are followed by two fully connected layers, but the last one outputs 28 classes (26 letters, "Space" and "Enter").

We use large 2-second fragments for classification. The center of the fragment encloses the keystroke samples identified by the keystroke detector. The size of the fragment significantly influences the results, with much lower accuracy for too small or too large windows. Not only does the network capture the power footprint of the keystroke, but it also recognizes the flanking keystrokes adjacent to the central one. The average time between two keystrokes is about 400-500 ms. Thus, the 2-second window includes up to four flanking keystrokes, all of which contribute to the classification of the one in the middle of the fragment.

The labeler uses an ensemble of 100 classification results obtained by moving a sliding window centered over the keystroke. Each input fragment for the CNN is obtained as follows: for every sample in the fragment produced by the keystroke detector, an input fragment is created for the labeler network by taking 2000 flanking samples, 1000 on each side. Since there are about 100 samples in the keystroke detector output, there are 100 inputs for the network per keystroke. The final outcome of the labeler is the softmax over all the outputs of all the 100 classification results.

Evaluation. We evaluate the accuracy by using a text with 2600 characters as a test set (using the classifier trained as in Section 6.2). Figure 9 shows the confusion matrix. There are three classes of characters: more likely to be recognized (10 out of 30, in particular "Space"), more likely to be misclassified (consistently confused, e.g. "b" with "n"), and could not be classified (e.g., "w").

These results show that the inference precision is about 10× better than random guess. However, they can be further improved if the attack's scope is narrowed. For example, one could use the confusion matrix (produced on some representative input) to take into account consistent misclassification errors, or boost the attack precision by using a language model [39]. We leave these for future work.

7.3 Other attacks

The general pipeline (Section 4) applies also to camera and incoming call attacks. They rely on the novelty detection stage that identifies the approximate fragment of the power trace that contains the event of interest, and then passes the fragment to the classifier. We use 2.5-seconds fragments in both attacks.

The attacks are based on a simple 1-Nearest Neighbors classifier just like the one used for the website inference. The input fragments, however, are not aligned with the signatures in the dictionary. Thus, the input is matched with the signatures by first identifying the best alignment to the signature (compute L^2 in a sliding window over the input signal and the signature, and find the one with the smallest distance), and then finding the smallest L^2 among all the signatures.

Attack	Experiment	Outcome
Web browsing	Downsampling	Precision above 50% down to 50Hz
	TOR Browsing	Cross-dictionary precision drops to 23%
	Cross-browser	Requires mixed dictionary
Keystroke inference	Downsampling	Average character accuracy 28% down to 100Hz
	Cross-user	Detection only
All attacks	Cross-phone (same make)	No degradation

Table 5. Summary of the robustness experiments.

8 Robustness analysis

We seek to determine the accuracy of the attacks under lower sampling rates and under different setups. We summarize the results in Table 5.

Cross-phone. We consider two cases: attack sensitivity to the specific device, and to the phone model. To evaluate the first case we use two Huawei Mate 9 phones with the same hardware and software configurations. We train the classifiers on the traces of the first phone and test them on the second phone. We find no noticeable accuracy degradation for any of the attacks.

However, the classifiers do not work across the phone models (e.g., Samsung Galaxy S4 and Note 4). This is an expected limitation of the power side channel attack. On the other hand, since different phone models usually have batteries of different geometry, the attacker may be able to bundle the correct classifiers to the battery for the specific model.

8.1 Web site inference

Downsampling. For every value of the sampling rate we downsample the traces in the dictionary and retrain the pipeline from scratch. We then downsample the test trace with 50% known webpages from the end-to-end attack and evaluate the precision. Surprisingly, the overall attack precision remains about the same down to 100Hz, and drops to 30% only at 25Hz. This sampling rate is qualitatively lower than the 7.8KHz minimum sampling rate reported previously [6]. The novelty detector is affected more than the classifier, which alone achieves a 46% success rate even at 25Hz. In other words, *the webpages remain distinguishable even at such a low sampling rate.*

Network conditions. We use the Orbot Android App to perform anonymous browsing via Tor, the anonymous onion routing network. We evaluate the attack precision in several scenarios.

First, we train on the traces of browsing via a regular network to classify browsing sessions performed via Tor. We measure the precision of the attack on three one-hour traces (100 browsing events per trace) with 50% white-listed webpages. As expected, the precision of the attack drops from 65% to 23%. We believe that this drop in precision is due to different network conditions via the Tor overlay, because only disabling the browser cache even without TOR reduces the precision to 34%.

Second, we retrain the classifiers on the traces collected while browsing via Tor (about 6.5 hours of browsing) and re-evaluate the attack precision. This time we achieve a 51% success rate.

Last, we keep the same Tor-trained pipeline as above, but now collect the test set traces while browsing via different overlay nodes. The precision is 43%.

We conclude that the *dynamic network conditions may reduce the accuracy of the attack.*

Different web browsers. We use native Android and Google Chrome browsers on a single phone, while training and testing the pipeline on different browsers. As expected, the attack's accuracy drops because the browsers' page rendering implementations differ substantially. However, the attack precision is restored by merging the dictionaries collected in different setups into a single combined dictionary. We merge the dictionaries collected for two different browsers, creating a combined dictionary with 2000 signatures (20 per each webpage, 10 for each browser) and retrain the pipeline. We achieve 59% for the end-to-end tests across the browsers, which is only 6% lower than the precision of the browser-specific pipeline.

8.2 Keystroke inference

We evaluate the average character accuracy of the end-to-end attack while reducing the sampling rate. We find that the accuracy does not change (about 36%) down to 500Hz, and even 100Hz is enough to obtain 28% accuracy, which is about 10 times higher than the random guess. However the accuracy drops afterwards, reaching 8% at 20Hz. We find that the per-character accuracy for all characters but the Space drops to almost zero, whereas the space character remains clearly distinguishable with about 60% accuracy.

Cross-user. We train the classifiers on one user and test to recognize the typing of another user. We find that the keystroke *detection* remains accurate *across users*, but the labeler does not work. The reasons for the user sensitivity of the neural network-based classifier are hard to identify, because the weights of neural networks are difficult to interpret. However, our conjecture is that the classifier relies on certain user-specific features, most likely the timing between the keystrokes. This conjecture is based on the observation that the Hidden Markov Model used for the keystroke inference by Song et al. [29] is also highly user-sensitive. This model is based on the inter-keystroke time. We conclude that the key inference techniques are suitable primarily for targeted attacks.

9 Exfiltration

We establish a covert channel between the battery and a remote attacker, by manipulating a software-visible charging state from the battery, and then retrieving it from JavaScript running in a browser via the HTML5 Battery Status API.

The Battery Status API has long raised privacy concerns [21, 22], which eventually led to its removal from several popular browsers, most notably Mozilla Firefox (from V.52). However, we confirm that it is still supported in the recent (May 2018) Chrome V.63, making our exfiltration mechanism broadly deployable.

The API exposes three parameters: time to full (dis)charge, battery level and charging state. Unfortunately, the charge level and time to charge values are not suitable for the covert channel. First, the charge level exposed to the Battery Status API does not reflect the actual battery charge; instead it is the outcome of an opaque vendor-dependent model. Thus, it is quite challenging to reliably manipulate the charge-related values from the battery, in particular without any feedback for calibration. Second, the Battery Status API sampling rate is limited to a single update every 30 seconds.

Interestingly, it appears that no sampling rate constraints are enforced on the battery charging state. We find that the JavaScript app in a browser and a native app both detect the charging state changes immediately.

The main challenge is thus to manipulate the charging state from the battery. There is no simple way to do so for a wired charger, because the state switching is not controlled from within the battery.

Instead, the wireless charging technology, increasingly used in modern phones, can be exploited for that purpose. A resonant inductive charger relies on two coupled inductors, with the primary coil in the *transmitter* (the charging pad) and the secondary coil in the *receiver* (the mobile device). The charging state changes when the presence of an active transmitter is detected by the receiver. Thus, we can control the charging state without a physical connector by placing the malicious transmitter's coil inside the battery.

We implement the transmitter using a standard resonant wireless charger and a control transistor for switching it on and off. This circuit is placed into the battery, and controlled by the internal in-battery controller, allowing bit-by-bit transmission from the battery. For the transfer we use a well-known self-clocking phase encoding – Manchester code [32], which encodes the bits using state transitions, and a UART protocol for byte-coding with 1 start bit, 8 data bits and 1 parity.

The receiver is a JavaScript that subscribes to Battery Status events. It decodes the data and sends it to the web server using AJAX.

9.1 Evaluation

We use Samsung Galaxy S4 equipped with a wireless power receiver extension module in the battery compartment. For the transmitter we add a gate transistor to a standard Qi charging pad to switch its on/of status by using an external microcontroller.

We measure the maximum frequency at which the charging state can be changed from the battery and can be observed by the JavaScript running in the browser. Since the browser does not seem to limit the update rate, the state change depends entirely on the phone's software and the charging pad state transition rate. We find that the time to detect the transition from not-charging to charging (T_{dc}) is 3.9 seconds, and the tran-

sition back to not-charging (T_{cd}) is 1.6 seconds. Thus, the actual transmission rate is about 0.17 bit/second.

To isolate the sources of the delays we measure the charging state transition time for a wired charger. Here we obtain about 3× faster transitions, with the possible bit rate of about 0.5 bit/second. We believe that the wireless charger can be further improved to reach similar rates.

10 Deconstructing the Power Side-Channel

We drill down into the mobile device to locate *the source of* the information leakage in the power channel. Understanding the real reasons why the attack is successful helps assess the opportunities for defenses against it.

10.1 Website inference

We seek to answer the question: *what is the contribution of the SoC and its individual components to the power channel information leakage?* The SoC is known to consume significant power in mobile devices [4, 13], but there are other potential sources, e.g, display. Clearly, different answers to this question dictate different approaches to the attack mitigation.

We need to measure the SoC power draw directly from the Power Management Integrated Circuit (PMIC). If the original attack on these traces yields high precision, it would imply that the SoC is the primary source of the information leakage. Obtaining such fine-grained measurements in a mobile phone is challenging without low-level access to PMIC. Instead, we perform the experiments on the TI OMAP5432 EVM [33] development board. The board includes a 1.5 GHz dual-core A15 multi-core processor, a dual core SGX544 graphics processor, and 2 GB of DDR3L RAM. These specifications are representative of mobile SoC hardware [13]. We connect to the built-in sense resistors to measure the PMIC, CPU, GPU, and DRAM power rails.

10.1.1 Results

We collect the power traces from the PMIC and the individual SoC components while browsing the Alexa top 100 websites. We only employ the classifier phase of the attack to determine the ability to distinguish between websites.

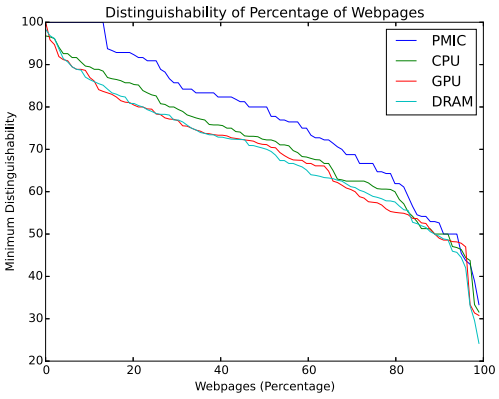

Fig. 10. Distribution of the website precision for attacks on CPU, GPU, DRAM and PMIC.

PMIC. Figure 10 presents the classification precision for the websites we study. More than 10% of the pages are distinguished 100% of the time and about 25% of them are distinguished with accuracy greater than 90%. The precision is even higher than that obtained from the phone in Section 7, which indicates that *the SoC is indeed the primary source of the information leakage.*

CPU,GPU,DRAM. The CPU, GPU, and DRAM are each individually vulnerable to the power side-channel attack. Figure 10 shows the classification accuracy for these three components separately. The accuracy of each of the channels is only 10% worse, with the attack on the CPU traces being the most accurate of the three.

We find that the CPU's power consumption contributes most significantly to the overall power side-channel. The Pearson correlation between the PMIC power consumption and each component is the highest for the CPU (r=0.94 – strongly correlated), while only 0.61 and 0.70 for the DRAM and GPU respectively. This motivates us to focus on the CPU power management as a possible way to protect against the attack.

10.2 Keystroke inference

The keystroke inference attack relies on the observation that touching a keyboard character on the phone's touchscreen results in a distinguishable power signature. Importantly, the power footprint of the soft keyboard keystrokes is distinguishable from other screen touches. Thus, the information leaks in the power signature may have several sources: capacitive touchscreen, text post-processing or the related software stack [9, 11].

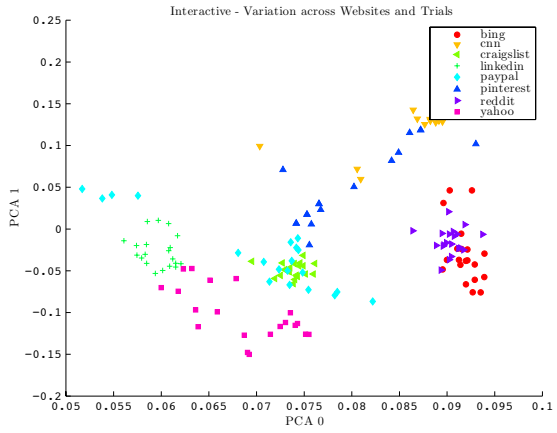

Fig. 11. PCA of the baseline attack

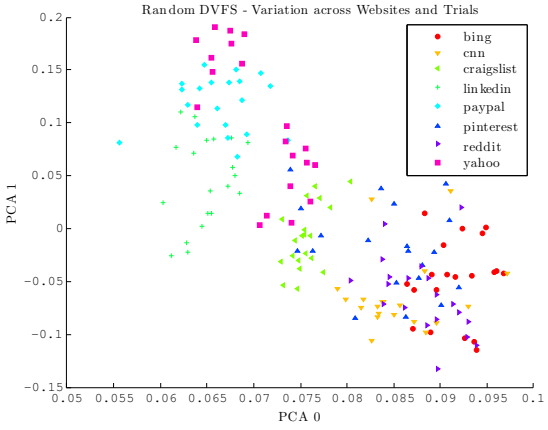

Fig. 12. PCA with randomized DVFS governor

To identify the real source, we connect a Bluetooth keyboard to the phone and measure the power while typing text with the soft keyboard displayed on the screen, with the predictive dictionary and autocorrection activated. In other words, the only observable difference is the use of a physical keyboard instead of the soft one. We observe no power signature for keystrokes in this experiment.

This result indicates that *the main source of the power signature is the touch screen*. This result is important when devising the defense against the attack in the next section.

11 Defense

We present a promising approach to protect against website inference attacks, but find that targeted defenses are required to protect against the other attacks. We show the defense against the keystroke attacks as a representative example.

Toward a DVFS-based defense. A promising approach is to leverage the DVFS mechanism to obfuscate the power trace. Such an approach has already proved successful on cryptoprocessors, but its efficacy for general-purpose workloads is unclear.

We made several attempts at constructing an effective DVFS-based defense. Two of the failed attempts are worth noting: fixed frequency and power normalization. Operating at a fixed frequency actually improves the accuracy of the attack by 9% (apparently reducing the noise from the dynamic behavior of DVFS itself). The second approach is to manipulate DVFS to maintain roughly constant power consumption when the CPU is idle (high frequency) vs. when it is fully loaded (minimal frequency). Here, the attack accuracy remained unchanged.

In contrast, a DVFS governor that randomly selects a DVFS state at one millisecond intervals reduces the attack success rate for website prediction to less than 5%. Figure 11 and Figure 12 show the Principle Component Analysis (PCA) of the power signatures for the subset of 16 websites obtained without and with the randomized DVFS governor respectively. We observe that the websites are less clustered with the deployed defense. While the results are encouraging, they are incomplete: the defense works only against one classifier – the 1-Nearest Neighbors (kNN) algorithm. Interestingly, the loosely clustered data (Figure 11), which aids the kNN also makes it sensitive to noise added by the random governor.

Targeted defenses. DVFS manipulation is not effective against attacks that exploit information leakage from peripherals. For example, it does not defend against the keystroke attack, because the information leakage occurs due to the touchscreen (§10). We note that prior defenses against the keystroke inference via software interrupt injection [26] are also ineffective here, because the keystroke power footprint differs from that of the software interrupt.

We resort to a targeted defense, whose purpose is to deliberately reduce the accuracy of the keystroke detector and therefore disable the attack. The idea is to inject random spikes into the power trace after each key release event. However, a defense-aware attacker may train the classifier to distinguish between the real and the artificial keystrokes. We empirically find that running a compute-heavy loop with a large randomly se-

lected number of iterations fools the keystroke detector, reducing the detection accuracy to zero.

This defense, however, shares the weakness of DVFS randomization: it is effective against a particular classifier, but may fail against others. Further, this defense works only for a particular hardware model, and does not apply to other attacks, e.g., camera and phone calls. These weaknesses motivate more research seeking a systematic approach for defending against malicious battery attacks.

12 Malicious battery hardware

We now discuss the characteristics of the hardware used to mount the attack. A complete implementation is outside the scope of this work, so we provide only best-effort estimates.

Power trace acquisition. Our power acquisition device is based on high-side current sensing using a shunt resistor between the positive battery terminal and the phone. Our early attempts to build the device were unsuccessful: the phone would not turn on due to high resistance of the shunt resister. Using smaller resistance (0.02 Ohm) does allow the phone to work, but it reduces the effective voltage drop and is more susceptible to noise, which requires the amplifier circuitry in place.

Trace storage for offline processing. There are many low-power off-the-shelf devices that provide all the required components in a compact form-factor that may fit under the battery sticker. For example, an off-the-shelf Arduino Nano with ATmega328 (with an ADC) with an amplifier and a storage controller is about 1.5 mm thick (likely even thinner on a custom board). This device consumes up to 50 mA at full activity, and 1-15 mA additionally consumed for writes to the SD drive. In total, its consumption is equivalent to that of a phone at rest (70 mA [4]). Importantly, it runs at 16 MHz, sufficient to execute activity and novelty detection at real time and write to storage at up to 650 KB/s.

Storage requirements. The storage requirements depend on the sampling rate and width. With the 200 Hz sampling rate, which results in negligible loss of precision, and 13 bits/sample we effectively use in the experiments, a continuous recording of an uncompressed trace over 1 month (about $2.6*10^6$ seconds) results in 0.8 GB/month, while requiring small write throughput of 325 bytes/second. Using compression and filtering the inactivity periods may reduce these requirements even further.

Online processing and exfiltration. The online scenario requires more powerful processors and uses our exfiltration technique (§9). For example, the online attack may run on a Texas Instruments Digital Signal Processor (DSP) TI C5504. This processor consumes less than 10 mA, and runs at 100 MHz, sufficient to perform complex computations, including Neural Network inference. In addition, it hosts 128MB DRAM that consumes 45 mA. This memory is large enough to hold a large power signature dictionary for the classifiers. The exfiltration coil is a sticker, with a small additional circuit. It is active only for exfiltration when the user is browsing a malicious website, and consumes about 8.6 J/bit.

To summarize, the low cost, simple construction, small size and low sample rate make the power measurement circuitry quite easy to integrate into a phone battery.

13 Conclusions

We introduce a novel *malicious battery* attack that stealthily records phone's power trace from the battery and successfully infers sensitive private information about the user. We demonstrate several novel attacks, evaluate them end-to-end on three popular phones and study their robustness under varying sampling rates and execution conditions. We then analyze the root causes for their success using low-level power measurements, devise a new data exfiltration technique from the battery directly to the remote server via a web browser, and discuss the steps towards attack mitigation.

Our findings clearly demonstrate that the malicious battery attack is powerful and feasible, it requires only cheap and compact components due to its low sampling rate requirements, and it is hard to mitigate completely, motivating further research into scalable and efficient defense mechanisms.

Acknowledgements

We are grateful to the anonymous reviewers, and to our shepherds Pepe Vila and Paolo Gasti for their constructive feedback. This research was partially supported by the Technion Center for Security Science and Technology (CSST), Hiroshi Fujiwara Cyber Security Research Center and the Israel Cyber Bureau.

References

[1] Anirudh Badam, Ranveer Chandra, Jon Dutra, Anthony Ferrese, Steve Hodges, Pan Hu, Julia Meinershagen, Thomas Moscibroda, Bodhi Priyantha, and Evangelia Skiani. 2015. Software Defined Batteries. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. ACM, New York, NY, USA, 215–229. https://doi.org/10.1145/2815400.2815429

[2] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *KDD workshop*, Vol. 10. Seattle, WA, 359–370.

[3] Bert den Boer, Kerstin Lemke, and Guntram Wicke. 2003. A DPA Attack Against the Modular Reduction Within a CRT Implementation of RSA. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '02)*. Springer-Verlag, London,UK, 228–243. http://dl.acm.org/citation.cfm?id=648255.752718

[4] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'10)*, Vol. 14. USENIX Association, Berkeley, CA, USA, 21–21. http://dl.acm.org/citation.cfm?id=1855840.1855861

[5] Yimin Chen, Xiaocong Jin, Jingchao Sun, Rui Zhang, and Yanchao Zhang. 2017. POWERFUL: Mobile app fingerprinting via power analysis. In *Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[6] Shane S. Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. 2013. Current Events: Identifying Webpages by Tapping the Electrical Outlet. In *Computer Security – ESORICS 2013*, Jason Crampton, Sushil Jajodia, and Keith Mayes (Eds.). Lecture Notes in Computer Science, Vol. 8134. Springer Berlin Heidelberg, 700–717. https://doi.org/10.1007/978-3-642-40203-6_39

[7] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. 2016. No pardon for the interruption: New inference attacks on android through interrupt timing analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 414–432.

[8] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1542–1552.

[9] Li Du. 2016. An Overview of Mobile Capacitive Touch Technologies Trends. *arXiv preprint arXiv:1612.08227* (2016).

[10] Denis Foo Kune and Yongdae Kim. 2010. Timing attacks on pin input devices. In *Proceedings of the 17th ACM conference on Computer and Communications Security*. ACM, 678–680.

[11] Shuo Gao, Jackson Lai, and Arokia Nathan. 2016. Fast Readout and Low Power Consumption in Capacitive Touch Screen Panel by Downsampling. *Journal of Display Technology* 12, 11 (2016), 1417–1422.

[12] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. 2016. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1626–1638.

[13] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. 2016. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 64–76.

[14] Jun Han, Emmanuel Owusu, Le T Nguyen, Adrian Perrig, and Joy Zhang. 2012. Accomplice: Location inference using accelerometers on smartphones. In *Fourth International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 1–9.

[15] Judith Horchert Jacob Appelbaum and Christian Stöcker. 2013. Der Spiegel. Shopping for Spy Gear: Catalog Advertises NSA Toolbox. https://nsa.gov1.info/dni/nsa-ant-catalog/, http://www.spiegel.de/international/world/catalog-reveals-nsa-has-back-doors-for-numerous-devices-a-940994.html. (2013).

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[17] Alexander Maxham. 2013. Android Headlines: Samsung Reaching 80 Million Galaxy S4 Sales. http://www.androidheadlines.com/2013/05/samsung-reaching-80-million-galaxy-s4-sales-in-2013.html. (2013).

[18] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. 2015. PowerSpy: Location Tracking Using Mobile Device Power Analysis. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 785–800. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/michalevsky

[19] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. 2012. Tapprints: Your Finger Taps Have Fingerprints. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*. ACM, New York, NY, USA, 323–336. https://doi.org/10.1145/2307636.2307666

[20] Roman Novak. 2002. SPA-Based Adaptive Chosen-Ciphertext Attack on RSA Implementation. In *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography (PKC '02)*. Springer-Verlag, London, UK, 252–262. http://dl.acm.org/citation.cfm?id=648119.761233

[21] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. 2016. The Leaking Battery. In *Revised Selected Papers of the 10th International Workshop on Data Privacy Management, and Security Assurance - Volume 9481*. Springer-Verlag New York, Inc., New York, NY, USA, 254–263.

[22] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. 2017. Battery Status Not Included: Assessing Privacy in Web Standards. In *3rd International Workshop on Privacy Engineering (IWPE'17)*.

[23] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. 2012. ACCessory: password inference using ac-

celerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM, 9–16.

[24] Power Flash 2016. Power Flash 1Cell SBS-compliant gauge IC for rechargeable smart battery pack applications. http://www.powerflash.com.tw/Product-1Cell.html. (2016).

[25] Morten Reintz. 2005. Atmel's ATmega406 AVR Microcontroller Provides Full Smart Battery and Battery Protection Functionality for 2 - 4 Li-ion Cells in a Single Chip. http://www.atmel.com/images/doc4083_mega406.pdf. (2005).

[26] Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, and Stefan Mangard. 2017. KeyDrown: Eliminating Keystroke Timing Side-Channel Attacks. *arXiv preprint arXiv:1706.06381* (2017).

[27] Omer Shwartz, Amir Cohen, Asaf Shabtai, and Yossi Oren. 2017. Shattered Trust: When Replacement Smartphone Components Attack. In *11th USENIX Workshop on Offensive Technologies, WOOT '17, Vancouver, BC, Canada, August 14-15, 2017*. USENIX Association. https://www.usenix.org/conference/woot17/workshop-program/presentation/shwartz

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

[29] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. 2001. Timing Analysis of Keystrokes and Timing Attacks on SSH.. In *USENIX Security Symposium*, Vol. 2001.

[30] Riccardo Spolaor, Laila Abudahi, Veelasha Moonsamy, Mauro Conti, and Radha Poovendran. 2017. No Free Charge Theorem: A Covert Channel via USB Charging Cable on Mobile Devices. In *International Conference on Applied Cryptography and Network Security*. Springer, 83–102.

[31] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. 2017. Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices. *IEEE Communications Surveys & Tutorials* (2017).

[32] Andrew Tanenbaum. 2002. *Computer Networks* (4th ed.). Prentice Hall Professional Technical Reference.

[33] Texas Instruments OMAP5432 Processor-based EVM 2013. OMAP5432 EVM System Reference Guide. http://www.ti.com/tool/OMAP5432-EVM. (2013).

[34] Kris Tiri and Ingrid Verbauwhede. 2005. Design Method for Constant Power Consumption of Differential Logic Circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1 (DATE '05)*. IEEE Computer Society, Washington, DC, USA, 628–633. https://doi.org/10.1109/DATE.2005.113

[35] Qinglong Wang, Amir Yahyavi, Bettina Kemme, and Wenbo He. 2015. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, 433–441.

[36] Zhi Xu, Kun Bai, and Sencun Zhu. 2012. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 113–124.

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

[38] Kehuan Zhang and XiaoFeng Wang. 2009. Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems. (2009), 17–32.

[39] Li Zhuang, Feng Zhou, and J Doug Tygar. 2009. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security (TISSEC)* 13, 1 (2009), 1–26.