Plug and Power: Fingerprinting USB Powered Peripherals via Power Side-channel

Riccardo Spolaor*, Hao Liu*, Federico Turrin†, Mauro Conti†‡, Xiuzhen Cheng*
* School of Computer Science and Technology, Shandong University, Qingdao, China.
† Department of Mathematics, University of Padua, Padua, Italy.
‡ Delft University of Technology, Delft, Netherlands.
Email: {rspolaor, xzcheng}@sdu.edu.cn, 202035137@mail.sdu.edu.cn, {turrin, conti}@math.unipd.it

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229048

***Abstract*—The literature and the news regularly report cases of exploiting Universal Serial Bus (USB) devices as attack tools for malware injections and private data exfiltration. To protect against such attacks, security researchers proposed different solutions to verify the identity of a USB device via side-channel information (e.g., timing or electromagnetic emission). However, such solutions often make strong assumptions on the measurement (e.g., electromagnetic interference-free area around the device), on a device's state (e.g., only at the boot or during specific actions), or are limited to one particular type of USB device (e.g., flash drive or input devices).**

In this paper, we present PowerID, a novel method to fingerprint USB peripherals based on their power consumption. PowerID analyzes the power traces from a peripheral to infer its identity and properties. We evaluate the effectiveness of our method on an extensive power trace dataset collected from 82 USB peripherals, including 35 models and 8 types. Our experimental results show that PowerID accurately recognizes a peripheral type, model, activity, and identity.

***Index Terms*—USB Security, Power Side-Channel, USB peripherals, Hardware fingerprinting.**

I. Introduction

Universal Serial Bus (USB) is the de-facto standard for the connections of a broad range of peripheral devices with higher speed transfer capability. The USB standard supports two main functions: data transfer (e.g., between USB guest and host) and power supply (e.g., smartphone charging). While these standards have been highly studied and improved over the years, little importance was given to their security [1], [2]. Indeed, the USB standard still lacks basic security practices such as encryption and authentication [2]. This aspect has exposed the USB ecosystem to many threats and exploitations [2].

On the one hand, the default trust on USB ports on a host device (e.g., workstation, public charging station, power bank) can be exploited by hackers to exfiltrate private information from USB devices [3], such as smartphones, tablets, and flash drives (i.e., host-to-guest attack). On the other hand, an attacker can disguise a malicious USB peripheral [4], [2] (i.e., guest-to-host attack) as a legitimate one to inject harmful commands (BadUSB, Mousejack, Rubber Ducky) deploy malware [5], [6], steal private user information (OMGCable, BadUSB2.0), spy on a user (tiny microphone/camera, cottonmouth, GSM spy bug), or destroy host's hardware (USBKill). Hence, malicious USB peripherals can cause severe harm to a host device if not promptly identified and blocked.

In recent years, several research works have investigated the feasibility of fingerprinting USB devices to protect host devices by relying on Physical Unclonable Functionss (PUFs) [7], or side-channels such as timing [8], [9] and electromagnetic emissions [10]. To the best of our knowledge, no work in the literature considers Power Side-Channel (PSC) information to fingerprint USB peripherals.

Using PSC information to fingerprint a USB peripheral has two main advantages. First, a USB peripheral's power traces are extremely difficult to replicate due to the complexity and specificity of its hardware components. Second, power traces do not retain information about the content of USB packets. Hence they preserve the confidentiality of the data exchanged between the peripheral and the host.

This paper presents PowerID, a framework that profiles USB peripherals from their power traces. PowerID leverages time series analysis and machine learning techniques to fingerprint the power trace related to a USB peripheral type (e.g., webcam, flash drive, or keyboard) and its specific actions (e.g., booting, writing on memory, or downloading via WiFi). Hence, users can rely on PowerID to uniquely fingerprint their peripherals and detect the connection of unauthorized devices or illicit actions. Since PowerID only leverages PSC information, it can be deployed on an external standalone device, i.e., it does not require additional information from or access to the host device. We develop PowerID framework and evaluate its effectiveness on a large-scale dataset composed of the power traces generated by distinct actions for a wide range of USB peripherals. Our experimental results demonstrate that PowerID recognizes with high accuracy the peripheral type, its specific model, and actions performed. Furthermore, we also show that PowerID can effectively detect attack tools disguised as a legitimate flash drive (e.g., BadUSB), thereby helping to protect host devices against such threats.

We can summarize our contributions as follows:

- We present PowerID, the first framework that leverages PSC analysis to fingerprint USB peripherals, infers their properties, and recognizes ongoing actions.
- Via an automated power traces collection system of our design, we collect a large dataset* with solid ground

*The dataset is available at https://doi.org/10.5281/zenodo.7467989

truth from 82 USB peripherals (8 types and 35 different models).
- We evaluate the performance of PowerID on the power traces dataset on different classification tasks. We show that PowerID can build robust fingerprints and recognize them with high accuracy.

II. Background and Threat Model

In this section, we provide useful concepts to understand the remainder of the paper. We briefly recall the USB power management system in Section II-A. In Section II-B, we present the threat model for PowerID and the related security scenarios. Then, we state the research questions we investigate in this work in Section II-C.

A. USB Power Management

The USB is the most popular standard for wired connections between peripheral devices (e.g., flash drive) and a host (e.g., a laptop) for exchanging data and powering. The USB standards define the unit load for the power draw of a USB device, i.e., $100mA$ and $150mA$ for USB 2.0 and 3.0, respectively [11]. When connected to a port, a device triggers a series of preliminary operations, including the initial handshake, device enumeration, and configuration. While only one unit load is initially provided by default, a device can request up to six unit loads in USB 3.0+ during the configuration for a maximum of $900mA$. Henceforth, we refer as state *Boot* to the above operations after a device is connected to a host. At the end of the state *Boot*, a USB device enters the state *Sleep* as no activity is ongoing. Upon the OS request or interrupt from the device, a device enters the state *On* to perform data transfer. Thus its power draw increases. Therefore, a device continuously switches between the states *On* and *Sleep* (i.e., life-cycle) until either removed from the port or turned off by the OS (i.e., *Sleep*).

The versatility of the USB standard enables the use of a broad range of peripherals categorized according to the type of data transfer: storage peripherals (e.g., flash drives) rely on bulk transfer since it guarantees delivery of extensive data but has a low priority over the bus (i.e., bandwidth); audio and video peripherals (e.g., webcams, microphones) use isochronous transfer that allows the stream of data with low latency but without guaranteeing delivery; and Human Interface Devices (HID) (e.g., mouse, keyboards) use an interrupt transfer with bounded latency. In this paper, we use the term *Type* for the purpose of a USB peripheral. Among the types, we consider the most common types of USB peripherals: flash storage drive (Fd), external hard drive (Hdd), WiFi and Bluetooth (Bt) network adapters, Microphone (Mic), Webcam (Wcam), Keyboard (Keyb), and Mouse. Specifically, manufacturers adopt their own or third parties hardware to produce USB peripherals. We refer to the combination of brand and model of a USB peripheral simply as *Model* (e.g., Kingston DT100 G3). As many peripherals share the same model, we define as *Device* the specific individual peripheral.

B. Threat Model

This paper investigates the feasibility of inferring coarse- and fine-grained information about a USB peripheral from its power traces measured at a USB port. Our system aims to protect the host device by identifying unauthorized USB peripherals, thus preventing subsequent attacks such as malware injection or private information exfiltration. In what follows, we describe possible use case scenarios, the attacker capabilities, and the PowerID preparation.

1) Use Case Scenarios: We conceive the threat model by considering two use case scenarios described in the following, where users and system administrators can rely on PowerID to enhance the security of the USB ecosystem.

End-user Personal Protection. Aside from malware-infected storage drives, USB attack tools disguised as flash drives or even concealed within USB cables are employed to perpetrate dangerous threats to user security and privacy (e.g., data exfiltration, command injection, credential theft). To protect from these attacks, users may want to assess the legitimacy of a connected-USB peripheral. Hence, users can deploy PowerID on a USB port to build power trace-based fingerprints for all their legitimate peripherals creating a whitelist of allowed personal devices.

Organization Assets Protection. An organization has a strict policy on the peripherals that its members are allowed to use. For example, the Stuxnet worm [5] has penetrated an air-gapped critical infrastructure via an infected USB storage drive. Hence, an organization's security team may enforce access control on its workstations to avoid potential attacks or human errors in connecting unauthorized USB peripherals. To this end, an organization can deploy PowerID to only allow the connection of USB peripherals with pre-approved characteristics (e.g., type, model) or permitted actions (e.g., read-only), alerting the security team whether anomalous power traces are detected.

2) Attacker Capabilities: We assume an attacker aims to compromise the host device of a user, an organization, or a critical infrastructure with a malicious USB peripheral. The attacker's objective may include delivering malware, exfiltrating sensitive information, or corrupting the host device. Therefore, the attacker replaces the legitimate USB peripheral with a compromised one (with the same appearance but concealing an attack tool) inducing the user to connect it to the host. Moreover, the attacker may obtain physical access to a user's host device (e.g., lunch-time attack) and attempt to connect its attack tool to the USB port.

3) PowerID Preparation: We assume that the adversary cannot interfere with power traces collection (e.g., sensor tampering) or compromise the model training phase (e.g., poisoning attack [12]) since such processes are crucial in the PowerID preparation. To obtain optimal fingerprints, we assume that the hardware settings during these processes are the same (or similar) as the ones of the final deployment (i.e., testing phase). Therefore, the device employed for the data collection is the same one used during the testing phase.

We believe this is a reasonable assumption since organizations typically purchase many identical host devices with the same hardware components. PowerID framework relies on an external standalone device to collect the power traces from an electric current sensor between the host and the peripheral under test (more details in Section III-A).

Traces processing and analysis can be done locally on such a device or remotely on a server (i.e., such a device streams the collected traces via networking technologies).

C. Goals of the Analyses

In our analyses, we assess whether we can infer information about a USB peripheral from the power traces analysis. In particular, the objective of our analyses is to answer the following questions:

① **Type**: Can we recognize the *type* of a USB peripheral during its states *Boot* and *On*?
② **Model**: Can we distinguish the specific *model* of a peripheral during its states *Boot* and *On*?
③ **Device**: Given peripherals of the same model, can we assess the identity of a specific *device*?
④ **Action**: Considering a peripheral in *On* state, can we recognize an ongoing *action* given a device type? E.g., reading from an Fd, or downloading from a WiFi adapter.
⑤ **Device via action**: Given an ongoing action for a type of peripheral, can we identify a specific *device*?
⑥ **Bad vs. Good**: Can we discriminate between malicious USB-based tools and legitimate peripherals?

III. PowerID System Design

In this section, we present our system and describe its components. In Figure 1, we provide an overview of PowerID. As a preliminary parameter, the *inference target* 1 defines the specific information about the connected peripheral as the target of the inference. Excluding the power traces acquisition 2, such parameter influences all the other components 3-6 and outcome 7 of our system. It is worth noticing that we can use the same power trace as input of several instances of our system with different inference targets. In what follows, we describe in detail the components of PowerID system.

A. Power Traces Acquisition

This component acquires the power traces of a connected peripheral via a sensor deployed between the port and such a peripheral (step 2). Such a sensor provides reliable measurements of the electric current supplied by the port. The resolution and the sampling rate of a sensor determine the quality of power traces acquired and their size. While a low resolution and sampling rate negatively affect the informativeness of a power trace, a high sampling rate requires more computational resources and time for processing. For this reason, it is important to find a trade-off between trace quality and the required processing and resources available. A sensor needs to be calibrated to provide readings within zero and maximum current provided by the considered USB port safely under the full-scale value to avoid saturation. As an additional

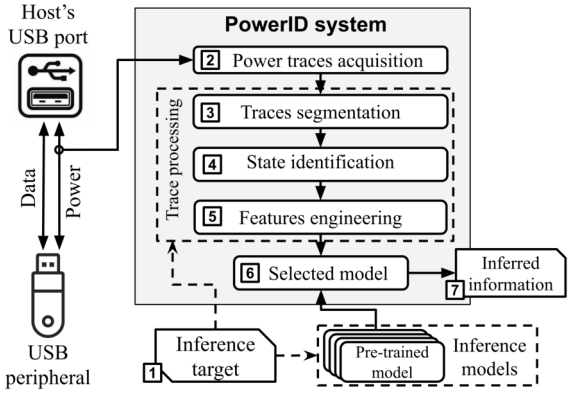


Fig. 1: The system design of PowerID.

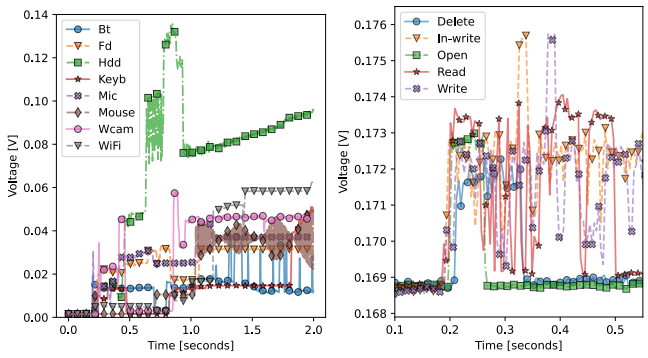
requirement, the sensor deployment should not affect the performance of the peripheral. With the above requirements in mind, we consider an Analog-to-Digital Converter (ADC) as a sensor that measures current in terms of voltage drop on a shunt resistor (see Section IV). Finally, this component delivers the power traces for further processing. In Figure 2 we report some examples of power traces collected from our experiments.

B. Traces Processing

We process the acquired power traces to obtain viable data for a machine learning-based model. The goal of this process is two-fold: preserving the information within power traces and identifying the current state of the peripheral. To attain this goal, we apply three methods: trace segmentation 3, state identification 4, and feature engineering 5. It is worth noticing that we apply the same process to obtain the datasets for both the model training and testing. However, while the state of a peripheral is known in the training data, during the testing, we identify the state with step 4.

1) Traces Segmentation: In this step, we divide the power traces into segments by applying a sliding window. Such a method considers two parameters: the *window duration* and *overlap ratio*. In selecting values for these parameters, we consider several insights obtained by observing the power traces of USB peripherals. Since some activities span for a brief time, a window with a long duration may produce segments that include an excessive amount of data unrelated to a state or action. Hence, such segments may not contain enough meaningful information for a considered inference target. The overlap allows us to obtain more segments for training our models, make them more robust to noise, and avoid overfitting. However, a big overlap can produce excessive segments and information redundancy, leading to high computational overhead in the processing phase. From observing the obtained traces, we set a duration window of 1 second with a 75% overlap.

2) State Identification: In our system, we focus on the states *Boot* and *On* of a USB peripheral. The analysis of segments from state *Boot* allows achieving the inference target



(a) State *Boot* for different types. (b) State *On* of a Flash drive.

Fig. 2: Example of power traces of states *Boot* and *On* for different USB peripherals.

1 within a few seconds from the connection of a peripheral to the monitored USB port. PowerID also relies on state *On* segments to infer all the considered target information, especially the ones related to the actions on a peripheral (i.e., ④ and ⑤). Due to low variability, we do not consider the power traces during the state *Sleep*. While identifying the starting time of a state *Boot* is trivial (i.e., change between open to close circuit), the identification of the transition time between states *Sleep* and *On* requires a refined approach since different peripherals produce heterogeneous power consumption in these states. Hence, we need a general approach independent of the considered peripheral (rather than setting a specific threshold for each USB peripheral type and model). To do this, we apply on groups of four consecutive segments a Changing Point Detection algorithm based on cumulative sum [13]. In particular, we consider a valid changing point if: (1) the next one does not occur within less than $250ms$, (2) there is at least 25% of value increase. We consider as the start of a state *On* the changing point identified by at least three segments among the four in the same group.

3) Features Engineering: We consider the segment of a power trace as a univariate time-series, i.e., sequential single data points over a constant time increment. Therefore, we apply the feature extraction method for time series provided by the *tsfresh* libraries [14]. Such libraries allow to extract 740 features from each segment, such as statistical features (e.g., mean, standard deviation, variance), linear trend, coefficients of Fast Fourier, and Continuous Wavelet Transform. The feature extraction depends on the inference target 1 and differs between the model training and testing phases. During the training phase, we select the k most significant features that effectively characterize the inference target. As a general approach, we aim to minimize the number k for two main reasons: (1) to not incur the curse of dimensionality and (2) to reduce the time and computational resources required by the feature extraction process. In the testing phase, we only extract the k meaningful features for the inference target.

C. Selected Model

For each inference target 1 considered, we train a classification model on the selected features for such a target. Upon the preliminary analyses, we select the Random Forest (RF) classifier as it achieves a higher performance among the other considered learners. While side-channel analyses via deep learning techniques [15], [16], [8], [17] can automate the feature selection process, the training of a deep neural network requires a huge number of examples and high computational and time resources due to the repeated training (i.e., epochs) for weights and parameters optimization. By performing the classification via non-deep learning-based techniques, we can pre-select the important features, thus reducing the complexity of the problem (i.e., the number of features to extract from the time series) and, consequently, the model size in memory. In the testing phase, we load the previously trained model (step 6) related to the inference target and test the previously unseen power trace segments. As a result of the classification, the model provides in the output the inferred information 7.

IV. Experimental Setup

In this section, we describe the implementation details of the power traces collection framework in Section IV-A and the dataset collection and processing setup in Section IV-B.

A. Power Traces Collection Framework

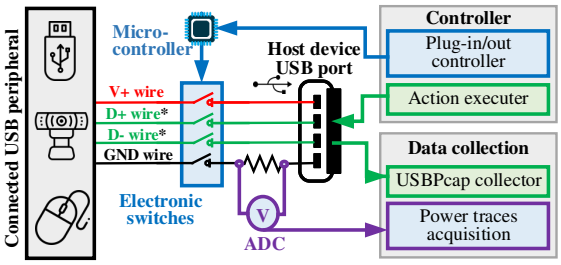
We design and implement a framework to collect the power traces for our analyses. In Figure 3a, we overview the logical components of our framework*, while in Figure 3b we show our experimental setup.

1) Power Traces Acquisition: In our experimental framework, we measure the electric current supplied by a USB port to a peripheral in terms of the voltage at the extremities of a shunt resistor (i.e., 0.01Ω) in a series of the GND (ground) wire of a USB extender cable. We measure such voltage with the ADC of a *National Instruments USB-6210* Data AcQuisition (DAQ) and acquire the measurements via the DAQExpress tool (depicted purple in Figure 3a). Although such ADC can acquire up to $125ksps$ (samples per second), we set the sampling rate to $10ksps$ to achieve a faster feature extraction.

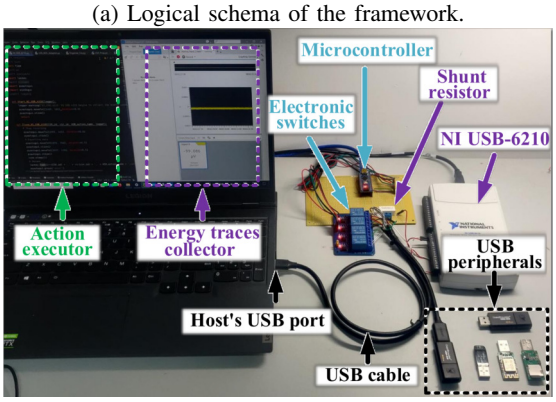
2) Actions Controller: For our analyses, collected power traces for states *Boot*, *On*, and *Sleep* of the USB peripherals. To automatize this process, we rely on two components: *Plug in/out controller* for *Boot* and *Action executor* for *On* and *Sleep*.

From the hardware perspective, the *Plug-in/Out controller* uses a series of electronic switches driven by a micro-controller (i.e., Arduino Nano). Data and power wires of the USB cable are connected to an electronic switch. From a software perspective, we can open or close the switches by sending the related command to the micro-controller. In particular, we implement such a function to replicate the

*For the sake of simplicity, we report a USB 2.0 pinout while our experimental setup fully supports the USB 3.1 standard



(a) Logical schema of the framework.



(b) Equipment setup.

Fig. 3: Framework for power traces collection.

insertion of a USB peripheral into a USB port (i.e., first connecting the power and then the data wires), thus triggering the state *Boot*.

Once the peripheral is connected to the host, the *Action executor* performs a list of actions (i.e., state *On*) according to the type of peripheral currently under test. In Table I, we report the list of considered actions for each peripheral type. For actions based on file transfer (e.g., Write, Download), the action executor involves a file randomly selected from a pool of files with assorted sizes (i.e., from 10MB to 200MB).

To obtain a reliable ground truth for our data, the scripts of the *Action controller* log the timestamp for each plugin/out and action. To assess whether a script is properly triggered and action on the peripheral, we also monitor the USB data traffic via USB sniffer (i.e., USBPcap collector).

B. Dataset Collection and Analyses Setup

We collected the power traces of the USB peripherals listed in Table II. In total, we collected 8 different device types, 35 different device models, and 82 unique devices. In total, we collect more than $6k$ traces for state *Boot* (i.e., around $43k$ segments) and more than $14k$ traces for state *On*. (i.e., around $132k$ segments). We obtain the power traces from the USB 3.1 port of a laptop Lenovo Legion AMD Ryzen 7 5800H 16-core CPU 3.2GHz with 16GB RAM running MS Windows 10 64-bit.

For the power traces processing and model training, we use a Desktop PC AMD Ryzen 9 5900X 12-core 3.7Ghz CPU with 64GB RAM running MS Windows 10 64-bit. As libraries, we utilize the *tsfresh* for the time series feature extraction, *Scikit-learn* for the machine learning model and evaluation metrics implementations, and *imblearn* to deal with dataset unbalance.

TABLE I: List of considered actions for device types.

Type	Action	Description
Flash Drive / Portable Hard Drive	Write	Transfer files from host to guest
	In-Write	Copy files locally guest to guest
	Open	Open a file inside guest device
	Read	Transfer files from guest to host
	Delete	Delete files from guest device
WiFi Adapter	Connect	Connect to a WiFi network
	Download	Download files via WiFi network
	Upload	Upload files via WiFi network
	Disconnect	Disconnect from a WiFi network
Bluetooth	Active	Transfer files via Bluetooth
Microphone	Active	Audio recording from the guest
Webcam	Active	Video acquisition from the guest
Mouse	Active	User activity (click, move, scroll)
Keyboard	Active	A user typing textual contents

TABLE II: List of the USB peripherals involved in our analyses (# indicates the number of individual peripherals).

Type	ID	Brand	Model	USB v.	#
Flash Drive	Fd1	Kingston	DT100 G3	3.2	6
	Fd2	Sandisk	3.2Gen1	3.2	6
	Fd3	Aigo	U310 pro	3.1	6
	Fd4	Aigo	U310	3.1	1
	Fd5	Kingston	DTKN	3.2	1
	Fd6	Kingston	MicroDuo3 G2	3.2	1
	Fd7	PNY	TA4-064	3.2	1
	Fd8	Sandisk	Ultra	3.2	1
Portable Hard Drive	Hdd1	WD	My Passport	3.1	2
	Hdd2	WD	Black P10	3.1	1
	Hdd3	Seagate	One Touch	3.2	1
	Hdd4	Toshiba	DTB420	3.0	1
WiFi Adapter	WiFi1	TP-Link	WN726N	2.0	6
	WiFi2	TENDA	U6 N300	2.0	6
	WiFi3	D-Link	DWA-171	2.0	3
	WiFi4	ASUS	USB-AC57	3.1	1
	WiFi5	ASUS	USB-AC68	3.0	1
	WiFi6	Ugreen	AC650 11ac	2.0	1
	WiFi7	Mercury	UD6H	2.0	1
Bluetooth Adapter	Bt1	Lenovo	BT5 LX1815	2.0	4
	Bt2	Ugreen	BT4 US192	2.0	1
	Bt2	TP-Link	TL-UB240	2.0	1
Microphone	Mic1	Ugreen	Desktop Mic.	1.1	4
	Mic2	Soaiy	L28	1.1	1
	Mic3	Depusheng	T7	1.1	1
Webcam	Wcam1	Logitech	C270	2.0	2
	Wcam2	Logitech	C920pro	2.0	1
	Wcam3	Philips	P506 HD	2.0	1
Mouse	Mouse1	Dell	MS116c	2.0	6
	Mouse2	Logitech	G102	2.0	2
	Mouse3	Logitech	M546	2.0	1
	Mouse4	Logitech	MX Master	2.0	1
Keyboard	Keyb1	Dell	KB216d	2.0	6
	Keyb2	Logitech	K845	2.0	2
	Keyb3	DURGOD	TAURUS K320	2.0	1

V. Experimental Evaluation

We analyze the performance of PowerID to answer the research questions in Section II-C. We select the RF classifier upon a preliminary comparison across several learners. In each analysis, we split the dataset into 80% training set and 20% testing set using a stratified approach to maintain the

TABLE III: List and description of the analyses. For each analysis, we mark the states and types involved in the considered analysis (* only data from models with at least four devices).

			States		Types			
Analysis	**Target**	**Approach**	**Boot**	**On**	**Fd**	**Hdd**	**WiFi**	**Other**
①	Type	Multiclass	✔	✔	✔	✔	✔	✔
②	Model	Multiclass	✔	✔	✔	✔	✔	✔
③	Device	Binary	✔	✔	✔*		✔*	✔*
④	Action	Multiclass		✔	✔			
				✔		✔		
				✔			✔	
⑤	Device	Binary		✔	✔			
				✔		✔		
				✔			✔	
⑥	Bad vs. good	Multiclass	✔	✔	✔	✔	✔	✔

same class proportions. By relying on a validation set (10% of the training set), we study the performance of a model trained by varying the number of features k. In particular, we select the k top features ranked by their ANOVA F-value. Since the analyses have different goals, the best feature set can change considerably. For this reason, we perform a different feature selection for each analysis. To mitigate the possible unbalancing in the training set, we employ SMOTE algorithm [18] to balance the elements in each class. We apply the above-described pipeline in all the considered analyses unless explicitly mentioned.

In Table III, we summarize the analyses we present in the remainder of this section. In particular, we report the states considered for every analysis and the device models employed for the classification. We refer as *multiclass* to a classification task involving more than two classes. Instead, we refer as *binary* to a specific binary classifier focusing on a single target class at the time (i.e., we apply a One-vs-All strategy). This second approach allows the model to create a decision bound around the target class to discriminate it with respect to all the other classes. In this type of classification, we report the aggregated results as the average of the binary classification of all the considered classes. Interested readers can find more details on the difference between these two approaches in [19].

To attain an open-world scenario in a multiclass approach, we consider an additional class *Other* composed of a random sample of segments (10% of the considered dataset) unrelated to the current analysis. Similarly, we attain an open-world condition for a binary approach by removing the 10% of the non-target classes from the training set but not from the testing set. By leaving out classes from the training set, we can evaluate the robustness of our models against unseen devices.

We evaluate our classification performance with several standard metrics: Precision (Pr), Recall (Re), F1-Score (F1), Geometric Mean (Gm), and Area Under the receiver operating characteristic Curve (AUC). For each analysis, we present the experimental results, highlight the meaningful insights and discuss the limitations.

TABLE IV: Results of the analysis ① for device type recognition in terms of F1, Pr, Re, and Gm.

Types	State = **Boot**, k=100				State = **On**, k=50			
	F1	**Pr**	**Re**	**Gm**	**F1**	**Pr**	**Re**	**Gm**
Fd	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99
Hdd	0.97	0.97	0.98	0.99	1.00	0.99	1.00	1.00
WiFi	0.99	1.00	0.98	0.99	1.00	1.00	0.99	1.00
Bt	0.93	0.93	0.93	0.96	0.98	0.97	1.00	1.00
Mic	0.95	0.93	0.96	0.98	0.98	0.97	1.00	1.00
Wcam	0.98	0.98	0.99	0.99	0.99	0.98	1.00	1.00
Mouse	0.95	0.94	0.95	0.97	0.99	0.99	1.00	1.00
Keyb	0.94	0.94	0.94	0.96	0.98	0.97	1.00	1.00
Avg.	0.96	0.96	0.96	0.98	0.99	0.98	1.00	1.00
Std.	0.02	0.02	0.02	0.01	0.01	0.01	0.00	0.00

A. Device Type Classification ①

In this analysis, we aim to classify the device type from the power traces during the states *Boot* and *On*. Considering the state *Boot*, we can assess the peripheral within a few seconds from its connection to a USB port. However, we also classify the type of a peripheral during its activity (i.e., state *On*) to continuously verify that its type would not change (e.g., an Fd turns into a spy camera/microphone after some time).

1) Method and Dataset: We collected power traces from peripherals and grouped them by device type, obtaining eight classes. We also consider an additional class *Other* where in the state *Boot* analysis corresponds to random traces from the state *On*, and in state *On* analysis corresponds to random traces in the state *Sleep*.

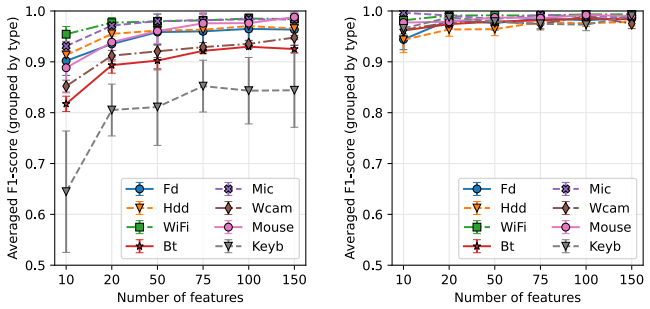
2) Results: We first analyzed the multiclass classification performance on the validation set, varying the number of features selected. By inspecting the Mouse and Keyb traces on the state *Boot*, we observe that most of them follow a sequence in terms of power draw: an initial peek at the connection (below 0.5 second), flat low, moderate, and stabilize in the state *Sleep*. Hence, we can assume that the model requires more information (features) and segments to classify them correctly. Hence, by selecting the best 100 features for the state *Boot*, we achieve the performance plateau with all the device types.

For state *On*, we can achieve high accuracy even with a small number of features as we reach the plateau with 50 features for all considered types. Considering the best feature sets for the two states separately, we report the classification result on the testing set in Table IV. PowerID models can discriminate with high accuracy between the device types for both the states *Boot* and *On*. Therefore, every device type has a unique fingerprint.

B. Device Model Classification ②

In this analysis, we delve a further level deep into the device identification by assessing whether the states *Boot* and *On* can discriminate the device model. Therefore, we perform a multiclass classification by considering as classes all the different device models in Table II.

1) Method and Dataset: We group power traces by the device model, obtaining the 35 different classes. We consider



(a) State *Boot*. (b) State *On*.

Fig. 4: Performance for the analysis ② varying the number of considered features.

an additional class *Other* that for the state *Boot* analysis corresponds to random traces from the state *On* while for state *On* analysis corresponds to random *Sleep* traces.

2) Results: We report in Figure 4 the F1 on the validation set, varying the feature number. The results indicate the average F1 and the standard deviation of the device models belonging to the same device type. Considering the state *Boot*, we can notice in Figure 4a that the performance plateau for every device model is reached at around 75 features. As in the previous analysis, we also note that the Keyb models are hard to classify when using information from a few features. This is again due to the quick handshake between the host and the device during the *Boot* phase. Regarding the state *On* in Figure 4b, the classification performance is high. Thus, we can conclude that this state presents a clear fingerprint for all device models.

By selecting the best 75 features for both states *Boot* and *On*, the results on the testing set underline that most of the device models present a unique power fingerprint, also among devices of the same type. However, the device model fingerprints from the state *On* perform better than the ones on state *Boot*. In particular, the devices which perform worst are the *Keyb3* and *Fd8*, from a visual inspection of the traces for such models, we observe that the state *Boot* lasts a short time (below 0.5 second), thus more difficult to fingerprint.

C. Authentication of Individual Device ③

In this analysis, we aim to discriminate individual devices of the same models. This fine-grained analysis detects whether a device has been tampered with or substituted.

1) Method and Dataset: We consider power traces of the device models with at least four individual devices, i.e., the device models with # ≥ 4 in Table II. For the same model devices, we perform a binary classification using one device as a target class at a time. Given a target device, we removed the segments of one random non-target device from the training set, and we added them to the testing set. We iterate this process for every device of a given model considering the states *Boot* and *On*.

2) Results: We analyzed the averaged F1 by device model varying the number of considered features. As expected, due to the high specificity of this analysis, we generally achieve lower performance than analyses ① and ②. For both states, we require at least 100 features to reach an F1 higher than 0.95 for most models. The detailed results of the testing set are reported in Table V. Confirming the results from the previous analysis, PowerID achieves an F1 higher than 0.9 for most models on state *On*, but it cannot correctly discriminate a few individual *Mouse1* and *Keyb1* devices for the state *Boot*. As a noticeable exception, the *WiFi1* model has the lowest score on the state *On*. Upon further inspection, we confirm our findings by observing that the traces of the action are very similar among the devices of such a model.

D. Actions Classification per Device Type ④

In this analysis, we focus on the state *On*, and we investigate whether we can infer type-specific actions across all models. After identifying the device type (analysis ①), PowerID aims to identify unexpected action a device performed (e.g., unauthorized file transfer).

1) Method and Dataset: We consider three device types of power traces: Fd, Hdd, and WiFi. We focus on these types because they have a broader set of actions to analyze (see Table I). For every device type, the class *Other* is composed of random segments of actions by the other types.

2) Results: Figure 5 reports the results of the analysis. In particular, Figure 5a shows that the actions of *WiFi* type have a clear fingerprint while Fd and Hdd require a higher number of features to reach an average F1 higher than 0.8. For a better understanding, in figures 5b, 5c, and 5d we report the confusion matrix with the 75 best features selected for Fd, Hdd, and WiFi, respectively. In figures 5b and 5c, we can observe that most of the miss-classification of Fd and Hdd are between *Write* and *In-Write* actions. This is probably because *In-Write* is derived by the combination of *Read* and *Write*. Therefore, *In-Write* generates power traces similar to the *Write* actions. Moreover, the Hdd type suffers from the *Other* class mainly due to the similarity with traces of the Fd type.

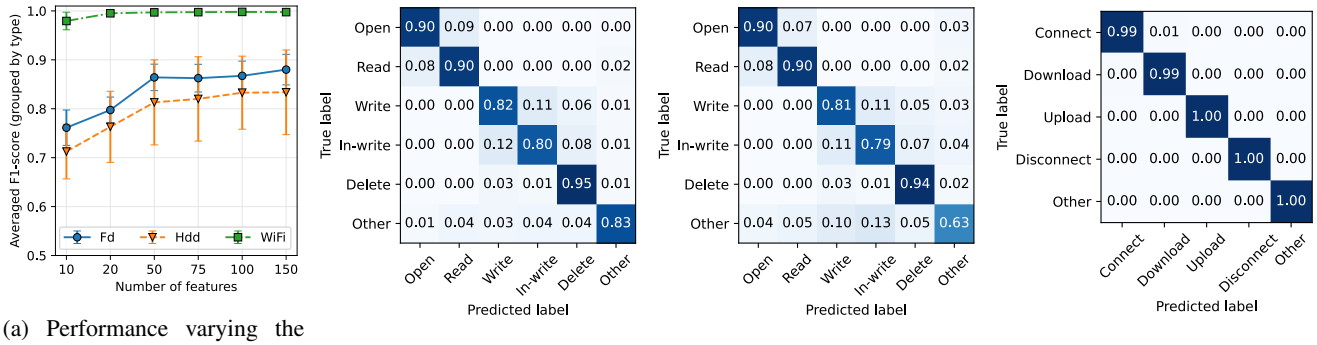
E. Individual Device by Actions ⑤

The previous results demonstrate that PowerID identifies with suitable accuracy types, models, the individual device of a specific model, and actions. Building on top of analyses ①, and ④, we investigate whether PowerID can discriminate individual devices from the actions they are performing.

1) Method and Dataset: Similarly to analysis ④, we focus on Fd, Hdd, and WiFi types. In particular, we consider *Read* and *Write* for Fd and Hdd, and *Download* and *Upload* for WiFi. These actions are the most commonly performed for legitimate and malicious purposes (e.g., data exfiltration, malware injection). We select the power traces of the actions from all device models for each type. Considering a class is the pair *(Device, Action)*, we obtain 46, 10, and 38 classes for Fd, Hdd, and WiFi, respectively. For each device type, we perform binary classification on such classes.

TABLE V: Results of the analysis ③ for device fingerprinting. For each row, we report the averaged score (and its standard deviation) across the devices of the same model.

Models	State = **Boot**, k=100					State = **On**, k=100				
	F1-score	**Precision**	**Recall**	**Gmean**	**AUC**	**F1-score**	**Precision**	**Recall**	**Gmean**	**AUC**
Fd1	0.97 ±0.04	0.97 ±0.05	0.97 ±0.03	0.98 ±0.02	1.00 ±0.00	0.98 ±0.01	0.98 ±0.02	0.98 ±0.02	0.99 ±0.01	1.00 ±0.00
Fd2	0.98 ±0.02	0.98 ±0.03	0.98 ±0.04	0.99 ±0.02	1.00 ±0.00	0.96 ±0.02	0.96 ±0.01	0.97 ±0.02	0.97 ±0.01	1.00 ±0.00
Fd3	0.98 ±0.04	1.00 ±0.00	0.96 ±0.08	0.98 ±0.04	1.00 ±0.00	0.91 ±0.07	0.91 ±0.07	0.91 ±0.07	0.95 ±0.04	1.00 ±0.00
WiFi1	0.98 ±0.01	0.99 ±0.01	0.98 ±0.03	0.99 ±0.01	1.00 ±0.00	0.79 ±0.02	0.72 ±0.02	0.88 ±0.05	0.90 ±0.01	0.97 ±0.02
WiFi2	0.99 ±0.01	0.99 ±0.01	0.99 ±0.01	0.99 ±0.00	1.00 ±0.00	0.97 ±0.02	0.97 ±0.03	0.98 ±0.02	0.98 ±0.02	1.00 ±0.00
Bt1	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00
Mic1	1.00 ±0.01	1.00 ±0.00	0.99 ±0.01	1.00 ±0.01	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00
Mouse1	0.82 ±0.12	0.79 ±0.15	0.87 ±0.10	0.90 ±0.07	0.96 ±0.03	0.95 ±0.05	0.95 ±0.06	0.96 ±0.05	0.97 ±0.03	1.00 ±0.01
Keyb1	0.90 ±0.12	0.92 ±0.07	0.89 ±0.17	0.92 ±0.10	0.98 ±0.03	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00	1.00 ±0.00
Avg.	0.96 ±0.06	0.96 ±0.06	0.96 ±0.05	0.97 ±0.03	0.99 ±0.01	0.95 ±0.06	0.94 ±0.08	0.96 ±0.04	0.97 ±0.03	1.00 ±0.01



(a) Performance varying the number of features.

(b) Confusion matrix for Fd.

(c) Confusion matrix for Hdd.

(d) Confusion matrix for WiFi.

Fig. 5: Results for analysis ④ for action identification per device type.

2) Results: We report the results in Figure 6. We represent with the low and high caps of error bars the best and worst scores among the individual devices, respectively. Overall, PowerID achieves good classification for all the types and actions, despite the high number of classes per type. In particular, we observe that in the case of Fd and Hdd, the different actions are distinguishable from one device to another. However, the WiFi type obtains slightly lower performance and higher variability. From a detailed analysis, we assess that some devices (of model WiFi1, WiFi2, and WiFi6) are misclassified due to similar behavior in several power trace segments. Despite the variability in the WiFi type, PowerID can correctly discriminate most of the devices. Hence, it is possible to fingerprint an individual device from its actions.

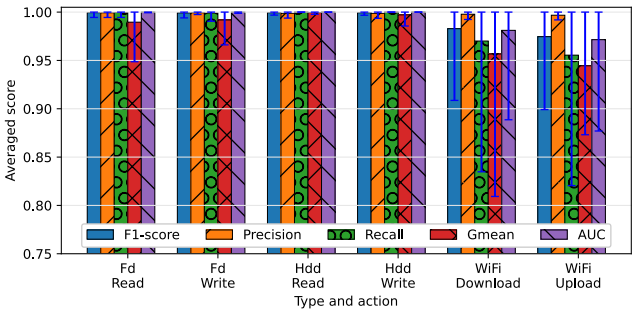


Fig. 6: Performance of analysis ⑤ for device fingerprinting on different actions.

F. Malicious Device Identification ⑥

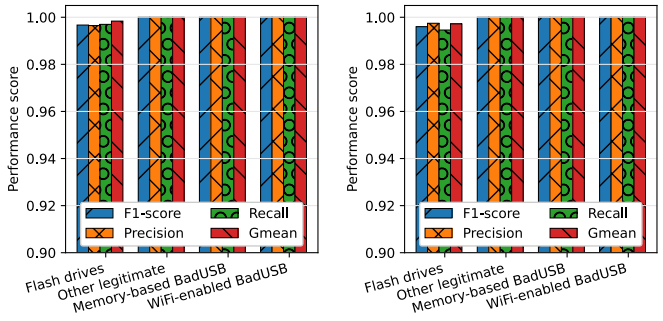
To assess whether it is possible to detect malicious devices from their power traces, we perform a multiclass classification between legitimate peripherals and BadUSB devices, focusing on states *Boot* and *On*.

1) Method and Dataset: In this analysis, we collect power traces from four BadUSB devices (two WiFi-enabled and two memory-based BadUSB devices). While collecting the traces, we run several common attacks, such as local (memory-based) and remote (WiFi-enabled) command injection on the command prompt (varying the types and number of commands) and WiFi scanning and connection. For the state *Boot*, we execute the attack at the device connection, while for state *On*, we delay its execution. The class *Other legitimate* is composed of traces from legitimate peripherals other than Fd type (e.g., Bt, Mic, WiFi). Moreover, we use the features set of the states *Boot* and *On* identified in ①.

2) Results: From the results in Figure 7, we observe that PowerID discriminates the BadUSB devices from Fd and other legitimate peripherals with perfect accuracy.

VI. Related Work

In recent years, attacks and countermeasures for the USB standard have been widely investigated [2], [4]. As an alternative to software-based approaches, researchers have considered the analysis of side-channels on the USB ecosystem [1]. We



(a) State *Boot*. (b) State *On*.

Fig. 7: Performance of analysis ⑥ on the discrimination between legitimate and malicious types.

are the first to propose a USB peripheral fingerprinting method that relies on the power side-channel. In what follows, we review and compare the work related to power and USB-based side channels.

1) Power Side-channel: Researchers in the literature propose various attack and profiling techniques via PSC considering different targets for information inference (e.g., passcode, browsing activities, encryption key), devices (e.g., PC, smartphones, embedded chips), and trace acquisition locations (i.e., local, vicinity, remote) [20]. In the vicinity settings, measuring the power traces at the source (e.g., wall-socket, USB port) is not invasive since it does not require any modification or tampering with the target device. Brighente et al. in [21] profile the Electric Vehicles charging based on the current exchange between vehicles and charging columns. Conti et al. in [22] use a wall-socket smart meter to obtain data on the power traces of laptops and identify authorized users. In the smartphone environment, Spolaor et al. [23] exploit the PSC to covertly exfiltrate user information encoded as power consumption bursts during the charging process of the victim's Android phone. Cronin et al. in [15] profile the dynamic content of the smartphone display to infer the PIN sequence by measuring the power consumption leaked via a charging cable. Under the same settings, Yang et al. in [24], [25] infer the user browsing activities in a smartphone via PSC information. Recent work also investigates the privacy leakage from wireless charging for smartphones. In particular, La Cour et al. in [16] profile the user browsing activities while Liu at al. in [17] infer additional private information such as passcode, keystrokes, and payment apps. Su et al. [26] assess that USB hubs expose a channel-to-channel crosstalk information leakage. In particular, they show how PSC of a USB port data lines are leaked from adjacent ports on the USB hub. However, no previous research considers the PSC analysis of USB peripherals (i.e., keyboards, webcams, flash drives), despite their prominent role in our everyday lives and their potential security threats [2].

2) USB Side-channels: In recent years, researchers have investigated the physical and logical side-channels of USB technology. Considering the physical side-channels, Belgarric et al. [27] and Genkin et al. in [28] use the ElectroMagnetic (EM) signal on the USB channel to recover the cryptographic key from a device connected via a USB cable. DeviceVeil [7] identifies individual USB devices by exploiting their PUF. This method achieves an accurate identification, but it requires a low-cost yet invasive hardware modification of individual USB peripherals. Differently, PowerID does not require any hardware or software modification of the peripherals under test. MAGNETO [10] is a framework to authenticate USB flash drives from their EM emissions. Despite sharing a similar goal, PowerID differs from this work in terms of side-channel considered, peripherals, and fingerprinting approach. In the first instance, MAGNETO and PowerID consider two different side-channels, EM and PSC respectively, making different assumptions in the threat model (see Section II-B). In particular, MAGNETO assumes that the peripheral under test is in an *electromagnetic safe zone*, which may be challenging to achieve in a realistic scenario. In the second instance, MAGNETO only focuses on operations during the booting (i.e., state *Boot*) for different models of USB flash drives only. Differently, PowerID considers a wider variety of USB peripherals, including multiple devices of the same model. Moreover, PowerID builds the fingerprints not only during a peripherals' state *Boot* but also in their activity (i.e., state *On*), which allows the identification of the ongoing actions (e.g., read/write, upload/download).

Other work considers USB logical side-channels (i.e., software-based). Monaco in [9] presents a fingerprint method for USB HIDs (e.g., keyboards, mouses, and touchscreens) based on clock timing and input events. Due to the peculiar nature of this side-channel, this work can only fingerprint HID peripherals, and it cannot be extended to other types of USB peripherals. Time-Print [8] authenticate USB flash drives by measuring USB packets' timing during a series of reading operations via a software tool on the host device. Unlike Time-Print, PowerID does not need to access the host device, and it aims to authenticate different types of peripherals (not limited to flash drives) based on their state *Boot* and from several actions (not only from reading operations), i.e., state *On*.

VII. Conclusion

This paper presents PowerID, a framework to fingerprint the USB peripherals based on their power consumption during different working conditions. Based on the fingerprints of authorized peripherals, PowerID can protect the host from USB-based threats by identifying unauthorized peripherals and detecting illicit actions. We extensively evaluated the performance of PowerID with an exhaustive power traces dataset composed of more than eighty unique devices spanning 35 models and 8 types. The results highlight that PowerID achieve a high accuracy in inferring peripheral type, model, activity, and identity. The authors have provided public access to their data at https://doi.org/10.5281/zenodo.7467989.

REFERENCES

[1] H. Liu, R. Spolaor, F. Turrin, R. Bonafede, and M. Conti, "USB powered devices: A survey of side-channel threats and countermeasures," *High-Confidence Computing*, vol. 1, no. 1, p. 100007, 2021.

[2] J. Tian, N. Scaife, D. Kumar, M. Bailey, A. Bates, and K. Butler, "Sok:" plug & pray" today–understanding USB insecurity in versions 1 through c," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 1032–1047.

[3] Larson, CNN Business. (2017) Please stop charging your phone in public ports. https://money.cnn.com/2017/02/15/technology/public-ports-charging-bad-stop/index.html. [Accessed: 07-06-2022].

[4] N. Nissim, R. Yahalom, and Y. Elovici, "USB-based attacks," *Computers & Security*, vol. 70, pp. 675–688, 2017.

[5] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[6] B. Bencsáth, G. Pék, L. Buttyán, and M. Felegyhazi, "The cousins of stuxnet: Duqu, flame, and gauss," *Future Internet*, vol. 4, no. 4, pp. 971–1003, 2012.

[7] K. Suzaki, Y. Hori, K. Kobara, and M. Mannan, "Deviceveil: Robust authentication for individual USB devices using physical unclonable functions," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 302–314.

[8] P. Cronin, X. Gao, H. Wang, and C. Cotton, "Time-print: Authenticating USB flash drives with novel timing fingerprints," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022.

[9] J. V. Monaco, "Device fingerprinting with peripheral timestamps," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022.

[10] O. A. Ibrahim, S. Sciancalepore, G. Oligeri, and R. D. Pietro, "Magneto: Fingerprinting USB flash drives via unintentional magnetic emissions," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 1, pp. 1–26, 2020.

[11] J. Axelson, *USB complete: the developer's guide*. Lakeview research LLC, 2015.

[12] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. Madison, WI, USA: Omnipress, 2012, p. 1467–1474.

[13] O. A. Grigg, V. Farewell, and D. Spiegelhalter, "Use of risk-adjusted cusum and rsprtcharts for monitoring in medical contexts," *Statistical methods in medical research*, vol. 12, no. 2, pp. 147–170, 2003.

[14] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[15] P. Cronin, X. Gao, C. Yang, and H. Wang, "Charger-Surfing: Exploiting a power line Side-Channel for smartphone information leakage," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 681–698.

[16] A. S. La Cour, K. K. Afridi, and G. E. Suh, "Wireless charging power side-channel attacks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 651–665.

[17] J. Liu, X. Zou, L. Zhao, Y. Tao, S. Hu, J. Han, and K. Ren, "Privacy leakage in wireless charging," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2022.

[18] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[19] Jason Brownlee. (2020) One-vs-Rest and One-vs-One for Multi-Class Classification. [Accessed: 07-06-2022]. [Online]. Available: https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

[20] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard, "Systematic classification of side-channel attacks: A case study for mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 465–488, 2017.

[21] A. Brighente, M. Conti, D. Donadel, and F. Turrin, "Evscout2.0: Electric vehicle profiling through charging profile," *ACM Transactions on Cyber-Physical Systems*, sep 2022.

[22] M. Conti, M. Nati, E. Rotundo, and R. Spolaor, "Mind the plug! laptop-user recognition through power consumption," in *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security*, ser. IoTPTS '16, 2016, p. 37–44.

[23] R. Spolaor, L. Abudahi, V. Moonsamy, M. Conti, and R. Poovendran, "No free charge theorem: A covert channel via USB charging cable on mobile devices," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 83–102.

[24] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, "On inferring browsing activity on smartphones via USB power analysis side-channel," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1056–1066, 2016.

[25] Q. Yang, P. Gasti, K. Balagani, Y. Li, and G. Zhou, "USB side-channel attack on tor," *Computer Networks*, vol. 141, pp. 57–66, 2018.

[26] Y. Su, D. Genkin, D. Ranasinghe, and Y. Yarom, "USB snooping made easy: Crosstalk leakage attacks on USB hubs," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1145–1161.

[27] P. Belgarric, P. A. Fouque, G. Macario-Rat, and M. Tibouchi, "Side-channel analysis of weierstrass and koblitz curve ecdsa on android smartphones," in *Cryptographers' Track at the RSA Conference*, 2016.

[28] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "ECDSA key extraction from mobile devices via nonintrusive physical side channels," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1626–1638.