Time-Print: Authenticating USB Flash Drives with Novel Timing Fingerprints

Patrick Cronin
University of Delaware
ptrick@udel.edu

Xing Gao
University of Delaware
xgao@udel.edu

Haining Wang
Virginia Tech
hnw@vt.edu

Chase Cotton
University of Delaware
ccotton@udel.edu

***Abstract*—Universal Serial Bus (USB) ports are a ubiquitous feature in computer systems and offer a cheap and efficient way to provide power and data connectivity between a host and peripheral devices. Even with the rise of cloud and off-site computing, USB has played a major role in enabling data transfer between devices. Its usage is especially prevalent in high-security environments where systems are 'air-gapped' and not connected to the Internet. However, recent research has demonstrated that USB is not nearly as secure as once thought, with different attacks showing that modified firmware on USB mass storage devices can compromise a host system. While many defenses have been proposed, they require user interaction, advanced hardware support (incompatible with legacy devices), or utilize device identifiers that can be subverted by an attacker. In this paper, we present Time-Print, a novel timing-based fingerprinting method, for identifying USB mass storage devices. We create a fingerprint by timing a series of read operations from different locations on a drive, as the timing variations are unique enough to identify individual USB devices. Time-Print is low overhead, completely software-based, and does not require any extra or specialized hardware. To validate the efficacy of Time-Print, we examine more than 40 USB flash drives and conduct experiments in multiple authentication scenarios. The experimental results show that Time-Print can (1) identify known/unknown brand/model USB devices with greater than 99.5% accuracy, (2) identify seen/unseen devices of the same brand/model with 95% accuracy, and (3) classify USB devices from the same brand/model with an average accuracy of 98.7%.**

I. Introduction

The Universal Serial Bus (USB) has been a ubiquitous and advanced peripheral connection standard for the past two decades. USB has standardized the expansion of computer functions by providing a means for connecting phones, cameras, projectors, and many more devices. Recent advancements in USB have increased data transfer speeds above 10 Gbps, making the USB mass storage device (flash drive) a popular method for moving data between systems. Especially, USB is commonly used in air-gapped systems where security policies prohibit data transfer via the Internet, such as military, government, and financial computing systems [13], [42], [44].

While USB has made the usage and development of various peripheral devices far simpler, it has recently been scrutinized for security issues [5], [2], [23], [30]. USB is an inherently trusting protocol, immediately beginning to set up and communicate with a peripheral device as soon as it is connected. This has many advantages, as users are not required to undertake a difficult setup process, but has recently been exploited by attackers to compromise host systems. The discovery of Stuxnet [2], Flame, and Gauss [36] has demonstrated that malware can be designed to spread via USB stick. Unwitting and curious employees might pick up dropped (infected) flash drives and plug them into their computers, allowing the malicious code on the drives to infect the hosts and then propagate across the network, wreaking havoc on the targeted industrial control systems. More recently, attackers have investigated the ability to modify the firmware of a USB device [23], [30] such that an outwardly appearing generic USB flash drive can act as an attacker-controlled, automated, mouse and keyboard. The behavior of the USB driver can also be utilized as a side-channel to fingerprint a host device and launch tailored drive-by attacks [5], [18]. While many defense mechanisms have been proposed, these techniques generally require user input [58], new advanced hardware capabilities [7], [55], or utilize features (device product ID, vendor ID, or serial number) that could be forged by an advanced attacker with modified firmware [1], [30].

In this paper, we propose a new device authentication method for accurately identifying USB mass storage devices. We reveal that read operations on a USB mass storage device contain enough timing variability to produce a unique fingerprint. To generate a USB mass storage device's fingerprint, we issue a series of read operations to the device, precisely record the device's response latency, and then convert this raw timing information to a statistical fingerprint. Based on this design rationale, we develop Time-Print, a software-based device authentication system. In Time-Print, we devise a process for transforming the raw timing data to a statistical fingerprint for each device. Given device fingerprints, Time-Print then leverages one-class classification via K-Means clustering and multi-class classification via neural networks for device identification. To the best of our knowledge, this is the first work to expose a timing variation within USB mass storage devices, which can be observed completely in software and be utilized to generate a unique fingerprint[1].

To validate the efficacy of Time-Print, we first provide evidence that statistical timing variations exist on a broad range of USB flash drives. Specifically, we gather fingerprints from more than 40 USB flash drives. Then we examine three common security scenarios assuming that attackers have different knowledge levels about the targeted victim from

[1]USB Type-C has provisions to identify device models [62] via a specialized key system; Time-Print does not make use of any specialized hardware and works on both legacy and new devices.

least to most: (1) identifying known/unknown devices with different models, (2) identifying seen/unseen devices within the same model, and (3) classifying individual devices within the same model. We demonstrate compelling accuracy for each case, greater than 99.5% identification accuracy between known/unknown devices with different brands and models, 95% identification accuracy between seen and unseen drives of the same model, and 98.7% accuracy in classifying individual devices of the same model.

We finally examine the robustness of Time-Print in multiple hardware configurations. We observe that Time-Print experiences a small accuracy degradation when measured on different USB ports, hubs, and host systems. We also examine the stability of Time-Print and present a strategy to make the fingerprints robust to write operations. Additionally, we investigate the authentication latency of Time-Print, demonstrating that while precise authentication can be achieved in 6-11 seconds, an accuracy greater than 94% can be achieved in about one second.

The major contributions of this work include:

- The first work to demonstrate the existence of a timing channel within USB mass storage devices, which can be utilized for device fingerprinting.
- The design and development of a completely software-based fingerprinting system, Time-Print, for authenticating USB mass storage devices without requiring additional hardware or burdensome user interaction.
- A thorough evaluation of more than 40 USB mass storage devices, showing that the ability to fingerprint with high accuracy is not dependent upon the device brand, protocol, or flash controller.

The remainder of this paper is organized as follows. Section II describes the threat model, including an attacker's capabilities, and provides a primer on the USB protocol, USB mass storage devices, and USB security threats/defenses. Section III demonstrates the existence of a fingerprintable timing channel within USB mass storage devices. Section IV details the method for generating and gathering a USB mass storage fingerprint. Section V presents the experimental setup for evaluation. Section VI evaluates the Time-Print system. Section VII examines the practicality of Time-Print under different use configurations. Section VIII surveys related work in USB security, device fingerprints, and device authentication. Finally, Section IX concludes the paper.

II. Threat Model and Background

This section presents the threat model and introduces various components of the new timing-based side-channel, including the USB protocol stack, USB mass storage devices, and current USB security.

A. Threat Model and Attacker Capabilities

The objective of this work is to highlight the applicability of a security primitive that can *physically* and reliably identify USB mass storage devices through a new timing-based side-channel. We consider a series of realistic scenarios, in which an entity attempts to either prevent its computing assets from engaging with unauthorized USB flash drives or better track the usage of flash drives inside an organization. The desired security level of a computing system inside the organization varies from the least (e.g., an open environment) to the highest (e.g., an 'air-gap' protection).

Under the lowest security level, we assume that attackers also have the least knowledge/privilege to launch an attack. For example, a computer at a reception desk or an open laboratory may have access to some assets on the organization's network and is in a high traffic area, where an attacker may be able to physically plug a malicious device into the temporarily unattended machine. However, compared to computing systems at the higher security level, it is less challenging (and with less motivation) to protect such computers at the lower security level. Moreover, the defense methodologies developed for a high-security system can be applied for the protection of a low-security system.

Therefore, the main focus of our work is on air-gapped systems that have the highest security level, such as computer systems in military, government, or financial organizations, which are frequently air-gapped and isolated from the Internet. Industrial control systems or life-critical systems (e.g., medical equipment) might also be air-gapped [42], [44]. While the air-gap is effective at thwarting the vast majority of outside attacks, it is very difficult to transfer data to and from an air-gapped system. To this end, USB mass storage devices offer an excellent, low-cost solution, but are not without their drawbacks. Attacks such as Stuxnet [2] were injected into target systems via USB, and recent research has demonstrated the creation of malicious USB devices which can negatively affect system security [5], [23], [30].

We then assume that attackers attempt to compromise the target air-gapped computer via USB drives. Attackers have the ability to design malicious USB devices so that once the USB handshake is completed, malicious scripts or activities can be executed on the host. According to the organization's security policies, system administrators only issue access to a few approved USB devices (i.e., insider devices) belonging to particular brands and models (e.g., SanDisk Cruzer Blade). Thus, a USB fingerprinting mechanism must be integrated into the host to accept/classify approved USB devices and reject other devices. For a specific air-gapped computer system, system administrators can train fingerprints for all approved devices. Also, they can pre-collect multiple devices from popular brands or models to augment the device authentication system with examples of unapproved drives.

With these settings in mind, we envision three typical scenarios as shown in Figure 1, in which Time-Print offers enhanced security benefits for device authentication. Note that Time-Print is designed to augment current USB security, and it can greatly assist existing USB security mechanisms such as GoodUSB [58] and USBFilter [59].

Scenario ❶: Attackers have no knowledge of the approved USB devices, and thus a random USB device could be connected to the target host. Such a random USB device likely

does not belong to one of the approved device models. Time-Print should thus reject any device whose model is not approved. In this minimal knowledge scenario, administrators can also prevent system infection from irresponsible employees that plug in non-approved devices (dropped devices) or computers in an open environment (reception computers).

Scenario ❷: Attackers (e.g., former employees who are aware of the security measure) know the brand and model of the approved USB devices and purchase one with the same brand and model. Time-Print should be able to reject unseen devices of the same brand/model.

Scenario ❸: Auditing user authentication. A system administrator should have the ability to identify specific devices that were issued to employees. For approved devices, different authorization levels might be assigned. In this case, the system administrator needs to audit which specific devices are connected to the target system to trace employee activities and detect data exfiltration attacks. Therefore, Time-Print should be able to classify all approved devices with high confidence.

Attacker Capabilities. We examine Time-Print against attackers at multiple levels. A weak attacker may simply attempt to plug a device into the victim system with little knowledge (e.g., Scenario 1). A stronger attacker may know the device model allowed at the victim side and attempt to connect a device of the same model (e.g., Scenario 2/3). The strongest attacker may be able to steal a legitimate device and attempt to replicate the physical fingerprint with an FPGA based system. While the FPGA based system may attempt to emulate legitimate firmware, the firmware for current USB flash drives is a closely guarded and proprietary secret. We do not consider a case in which an attacker is able to significantly modify the firmware of a (stolen) legitimate device. In addition, we also must exclude authorized users who attempt to maliciously harm their own computing systems. This is a reasonable assumption as authorized users who have privileges to access any system resources likely have little need for mounting such a complicated USB attack.

Defender Preparations. To use Time-Print, defenders (e.g., system administrators) should first have a security policy for limiting the employee usage of USB devices to specific models. Then, they need to gather fingerprint samples for their legitimate devices to enroll them into Time-Print beforehand.

B. USB 2.0 Versus 3.0

The USB standard consists of software and driver specifications that control the communication between two devices and has undergone several revisions. One major revision of the protocol, USB 2.0 [17], enables high data speeds (e.g., the High-Speed specification of 480 Mbit/s), and adds support for diverse peripheral devices including cameras, network adapters, Bluetooth, etc. The later introduced USB 3.0 [25] standard offers an increased 5 Gbit/s data rate and additional support for new types of devices. Also, USB 3.0 devices are backward compatible with USB 2.0 ports, but at 2.0's speed. USB 3.1 [26] further increases the data transfer rate to 10 Gbit/s with a modified power specification that increases the maximum power

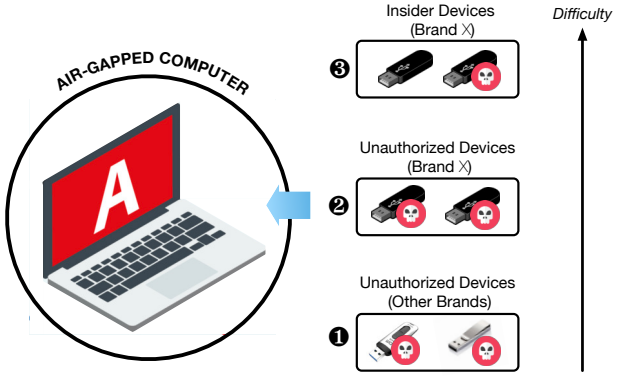

Fig. 1: Three security scenarios of USB fingerprinting for device authentication.

delivery to 100W [3]. In this paper, we focus on USB devices with standards 2.0, 3.0, and 3.1.

C. USB Mass Storage Devices and Flash Storage Controllers

USB mass storage devices are a form of removable storage media which allow a user to transfer files between a host and the device. As a recognized device class [21], mass storage devices follow a well-defined process when connected to a host. The host queries the device to discover its class code. Upon determining that it is a mass storage device, the host launches an instance (on Linux host systems) of the `usb-storage` driver. The driver scans the device, determining its file system, and launches the appropriate file system drivers.

To enable the communication between a device and the host, each USB mass storage device contains a microprocessor(s) that handles communications and manages the flash storage of the device. Flash storage is generally made up of many blocks. As flash has a limited write endurance and is usually designed in such a way that individual bits cannot be selectively cleared, the flash controller typically conducts a series of operations to modify the stored data in the flash medium. It first locates a new unused block and copies the data from the old block to the new block while incorporating any data changes. The flash controller then marks the old block as dirty, and eventually reclaims these dirty blocks as part of the garbage collection process. The controller (as the 'flash translation layer') maintains the mapping information between logical addresses (addresses used by the host system to access files) and the physical addresses of the actual pages, and the frequent remapping of blocks is an invisible process to the host system. Thus, the time required for the USB mass storage device to access large chunks of data is potentially unique and suitable to fingerprint the device.

D. USB Security

With its rapid adoption, USB has also become a popular target for attackers. Previous studies have shown that users are likely to plug in devices that they find on the ground [29], [53], [61], especially those modified to look 'official' (e.g., contain a government logo) [51]. Meanwhile, researchers have also proposed numerous defenses, ranging from firewall and

permissions systems [4], [58], [59] to device fingerprinting [27]. Many of these systems rely upon the reported device descriptors (e.g. product/vendor ID, serial number) [1], [8] which can be modified by a skilled attacker [23], [30]. Magneto [27] attempts to identify USB devices via electromagnetic fingerprinting of the microcontroller within the USB mass storage device, which is hard for attackers to manipulate. However, the system required expensive, bulky, and highly sensitive spectrum analyzers and EM probes to identify devices. Instead, in this work, we uncover a new timing channel within USB mass storage devices and require no extra equipment to uniquely identify devices.

III. Timing Side-Channel Exploration

USB mass storage devices are sophisticated systems that contain at least one microprocessor (i.e., flash storage controller), some form of embedded firmware, and one or more flash memory devices. The microprocessor(s) is utilized to maintain the flash translation layer and the flash endurance (via wear leveling) and to communicate with a host computer. Whenever the USB mass storage device connects to the host, a series of transactions provide the host with information about its size, capabilities, name, partition table, etc. For those transactions, individual physical devices may demonstrate small variations (e.g., timing variations) within a tolerance boundary that does not affect normal operations. One common method for observing these variations is through unintentional electromagnetic emissions [14], [15], [16], [18], [27].

While prior works have demonstrated that the USB device enumeration process can be used to identify individual host computers / OSes [5], [18], [37], we attempt to explore a timing channel to accurately identify USB devices. In particular, this work searches for observable timing differences between the interactions of a mass storage device and its host. If the flash controller of one device can respond faster or slower than that of a different device, it is possible that this variation can be used to identify a device. Furthermore, if a large chunk of data is requested from the device, the flash translation layer may access multiple locations to return all of the data at once. The time taken for this action (e.g., consult translation table, access one or multiple flash blocks within the device, coalesce data, respond to host) may also create observable timing differences.

A. Motivation of Time-Print

Previous works [5], [18], [37] have demonstrated that the USB handshake and enumeration process can leak information about the host, including the host's operating system (different command sequence) or the host itself (timing differences between packets). We first attempt to check whether such a handshake and enumeration process can also generate a stable fingerprint for USB devices.

Within the Linux operating system, this handshake entails the loading of a series of drivers, each providing more specialized functionality to the USB device. Once the device is initially connected to the host system, the USB core driver accesses the device and requests its descriptors. The device responds with its descriptors and identifies its class (e.g., human interface device, mass storage, etc.) A device object is created, and the specific class driver is then instantiated. In the case of a USB mass storage device, the USB storage driver is initiated, and the USB storage driver probes the device via its communication interface, the Small Computer System Interface (SCSI). The host utilizes SCSI commands to probe the filesystem, the appropriate filesystem driver is then loaded depending on format (e.g. FAT, exFAT, NTFS, ext4, etc.), and the drive is finally mounted and enumerated. With the drive handshake completed, the drive remains idle until the user opens the drive to access it.

We utilize the `usbmon` [67] driver within Linux and the `Wireshark` [56] program to capture and analyze the raw packet transmissions during the device enumeration and mount process. We find that the behaviors of packet transmissions between similar devices do not vary significantly enough to create a unique profile. In addition, the file contents of the same device greatly influence the behaviors of the device enumeration process, such as addresses, sizes, and the number of packets. Therefore, the device enumeration process cannot be leveraged to generate a reliable fingerprint.

B. Creation of a Reliable Fingerprint

To remedy this issue, we seek a new approach for creating a reliable fingerprint. While the timing of USB setup packets does not seem to provide fingerprintable information, the interfacing with the flash controller can. Each time the host system requests data from a USB device, the flash controller must access the flash translation layer, determining and accessing the location of the block (or blocks if the files are fragmented across multiple physical locations). It then coalesces those areas into USB packets and sends them to the host. Our intuition is that this access time varies based upon the locations of the blocks on a device as well as the size of a read. To examine whether this assumption is valid, we issue a known series of read requests of different sizes and locations via SCSI commands on the device. By recording the timestamp for each read action, we attempt to construct a statistical fingerprint for the timing characteristics of each device. We utilize the `sg_read` utility [19] to achieve low-level control of the read commands. Each read sets the Direct IO (DIO), Disable Page Out (DPO), and Force Unit Access (FUA) flags of the `sg_read` utility to '1'. This combination of flags forces the system to access the USB drive with each read and disallows the operating system from utilizing cached read data. Especially, the DPO flag forces the USB device to fetch the read from the physical media and keeps the flash controller from responding with cached reads. This flag combination is necessary to ensure that each read physically probes the specified flash blocks (allowing for true timing values to be gathered), instead of simply reading cached data.

C. Preliminary Classification

To investigate whether a timing fingerprint might be possible, we conduct preliminary experiments by gathering timing

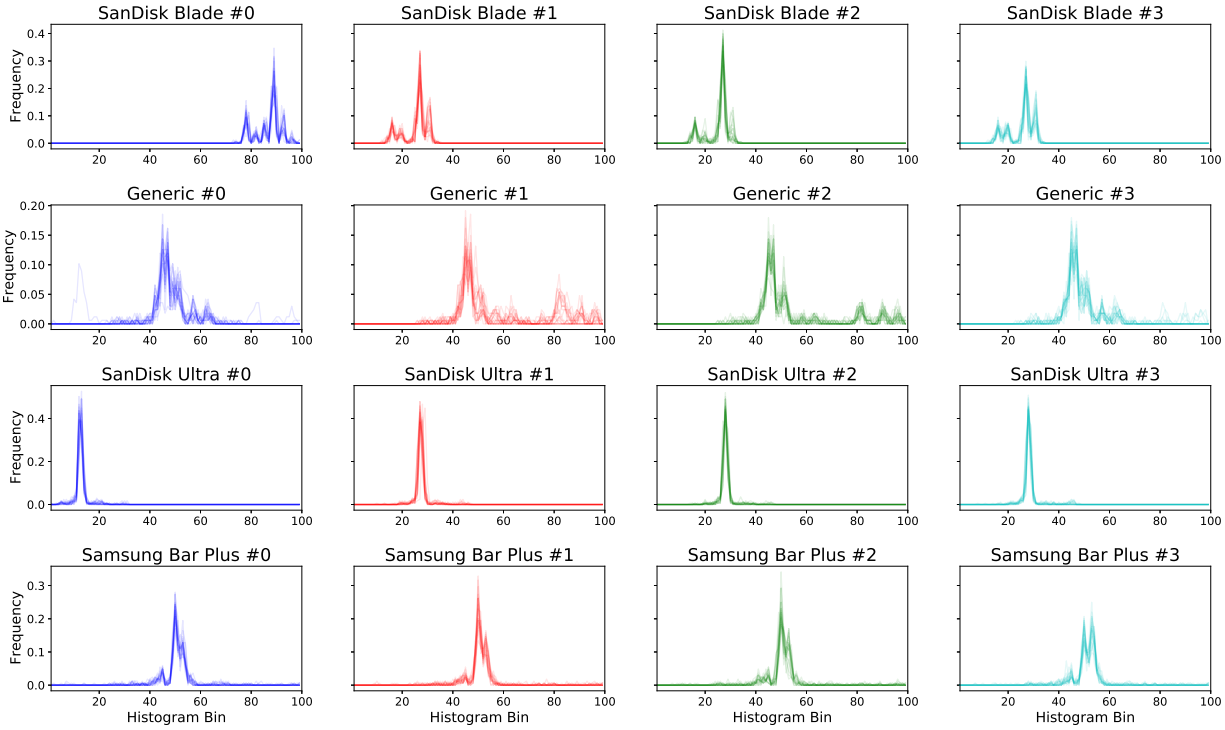

Fig. 2: Histograms of read timings for 16 different USB mass storage drives. Each plot contains 20 different samples.

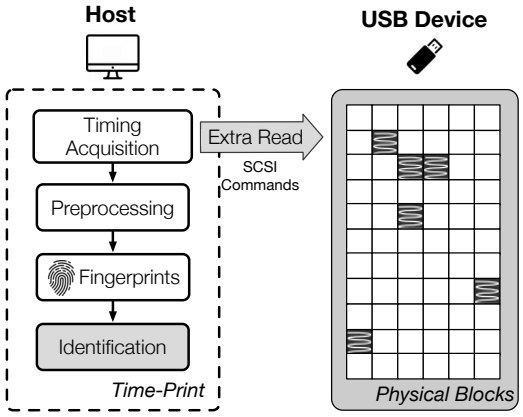

Fig. 3: The design of Time-Print.

readings from 16 different devices: 4 devices for each of 4 different models (i.e., a generic device found on Amazon, SanDisk Cruzer Blade, SanDisk Ultra, and Samsung Bar Plus). A histogram of these readings is presented in Figure 2. Each graph contains the histograms of 20 separate readings. The high overlap between readings implies that the timing measurement is stable from reading to reading, and thus may be a good candidate for fingerprinting. Visual inspection demonstrates that different brand/model devices exhibit different timing characteristics, indicating that read timings will enable us to differentiate devices with different models. Further inspection of the variations among devices of the same model shows that some clear differences still exist. For example, SanDisk Blade 1 and SanDisk Blade 3 in the first row demonstrate differently shaped distributions. Thus, the preliminary results motivate us to develop a timing-based device authentication mechanism.

IV. Time-Print Design

In this section, we detail the design and implementation of Time-Print and describe how Time-Print generates device fingerprints. In general, Time-Print extends the USB driver to generate a number of extra reads on randomly chosen blocks on USB devices via the SCSI commands (as shown in Figure 3) and then measures the timing information of these read operations. The process of Time-Print consists of four steps, namely, (1) performing precise timing measurements, (2) exercising the USB flash drive to generate a timing profile, (3) preprocessing the timing profile, and (4) conducting classification based on the timing profile for device acceptance/rejection.

A. Performing Precise Timing Measurements

As shown in Figure 3, Time-Print enables the fingerprinting technique within the driver using SCSI commands. Such a design allows the fingerprint data to be acquired before the device is fully connected to the host system (thus allowing for rejection if the device is deemed unrecognized). Also, the driver has visibility into every packet exchanged between the device and the host with minimal delay, which reduces the overhead and latency for the authentication process while simultaneously increasing the precision of the timing measurements.

The USB mass storage driver and the USB SCSI command sequence maintain a complex series of objects within the Linux operating system to control the command and data transactions communicated between the host and peripherals. Every data

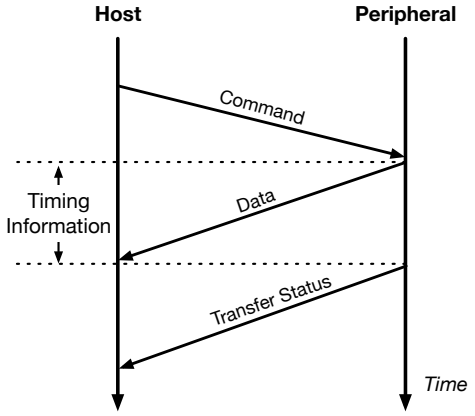

Fig. 4: A USB SCSI command sequence.

read consists of three parts, as visually presented in Figure 4: (1) the host issues a read command to the device, which specifies the size and location of the data to be read; (2) the peripheral responds with the requested data; (3) the peripheral responds with a status packet to indicate that the transfer is either successful or unsuccessful.

Within the USB driver, two different methods control these three transactions. The `usb_store_msg_common` method transfers the command packet and receives the status packet, while the `usb_stor_bulk_transfer_sglist` function receives the actual data from the device. To perform precise timing measurements of these transactions, Time-Print leverages a low overhead and high granularity timing source, the CPU timestamp counter (TSC), which is a monotonic 64-bit register present in all recent x86 processors. While initially designed to count at the clock speed of the CPU, most recent systems implement a 'constant TSC', which ticks at a set frequency regardless of the actual CPU speed. This feature enables Time-Print to precisely time the data transmission phase, regardless of the underlying CPU frequency. We utilize the built-in kernel function `rdtsc()` both before and after each transaction to record the precise amount of time it takes for the execution of each interaction.

With the collected timing information, Time-Print further integrates a low-overhead storage and reporting component for this timing information. This component modifies the USB driver to maintain a continuous stream of timing information for the drive. Specifically, we augment the `us_data` structure present in the USB storage header to contain arrays to keep track of command opcode, size, address, and TSC value for each transaction.

To transfer the timing values and record them (for prototype purposes), we implement a character device within the USB storage driver to transfer the timing information to the userspace for further processing. Since accessing the TSC is designed to be a low overhead function, the induced overhead is negligible (more discussion on the overhead is presented in Section VII). To ensure minimal performance impact, once a device has been approved, the timing and storage functionality can be disabled.

B. Exercising the USB Flash Drive

As discussed in Section III-A, it is difficult to build a reliable timing-based fingerprint based on the information leaked from the USB handshake and enumeration process, due to its variable nature. Instead, we develop a common test pattern that can be applied to any USB device. In particular, we generate a script with a random pattern of reads in different sizes from different offsets within the drive. The script is executed whenever a new USB device is detected by the host system. This procedure ensures a consistent number of reads from different locations on the drive allowing for the creation of a statistical, timing-based fingerprint. Meanwhile, reading from multiple locations with different sizes is necessary as it provides a better chance of generating a unique fingerprint for the flash drive. According to Micron [39], the NAND flash blocks built into a USB flash drive are at least 128KB, while each logical block address that can be accessed by the host system corresponds to a 512-byte chunk. As the logical to physical mapping is opaque to the user, it is challenging to know whether a large read from a specific location involves any accesses to multiple contiguous flash blocks, multiple blocks in different locations, or only a single block. By attempting to generate as many different types of accesses as possible, Time-Print can better extract the subtle timing differences caused by those accesses.

C. Preprocessing Timing Values

As shown in Figure 4, there are three packets exchanged between the host and peripheral: the original command, the responding data packet, and the transfer status. We need to capture and record the timing values for each packet from the host's perspective. Specifically, a timestamp is recorded upon the entry and exit of each of the two functions listed above. Each timestamp also includes the following meta-data: command opcode, the size of the packet, and the offset the data is coming from. The preprocessing step of Time-Print filters any commands that are not read commands from the recording, and searches for the beginning of the commands from the read script to discount any packets that are issued as part of the drive enumeration. As the goal of the fingerprint system is to focus specifically on the time it takes for the drive to access blocks of the USB device, not the timing between packets, we calculate the time latency between when the host finishes sending the command packet and when the host finishes receiving the data response packet from the drive.

The next step is to organize this raw timing information, which contains timing data from a multitude of locations and sizes. We group them into separate bins where each contains one size and address offset. Grouping the timing results by read size and offset ensures that each timing sample within a group corresponds to a single action or group of actions within the drive, allowing for meaningful statistical analysis.

D. Classification

With the timing information grouped by size and offset, we can leverage features and machine learning techniques to create a fingerprint for each device. Based on the trained

Device Manufacturer	Device Name	Size	Flash Controller	Number of Devices	USB Protocol
SanDisk	Cruzer Blade	8GB	SanDisk	10	USB 2.0
Generic	General UDisk	4GB	ChipsBank CBM2199S	10	USB 3.0
SanDisk	Ultra	16GB	SanDisk	10	USB 3.0
Samsung	BAR Plus	32GB	Unknown	4	USB 3.1
PNY	USB 3.0 FD	32GB	Innostor IS902E A1	1	USB 3.0
Kingston	DataTraveler G4	32GB	SSS 6131	1	USB 3.0
Kingston	DataTraveler SE9	64GB	Phison PS2309	1	USB 3.0
PNY	Elite-X Fit	64GB	Phison PS2309	1	USB 3.1
SMI	USB Disk	64GB	Silicon Motion SM3269 AB	1	USB 3.0
SMI	USB Disk	64GB	Silicon Motion SM3267 AE	1	USB 3.0
SanDisk	Cruzer Switch	8GB	SanDisk	1	USB 2.0
SanDisk	Cruzer Glide	16GB	SanDisk	2	USB 2.0

TABLE I: USB mass storage devices utilized in the evaluation of Time-Print.

fingerprints, Time-Print can reject or accept devices. For the different security scenarios mentioned in the threat model, Time-Print uses different algorithms for better performance. Section VI further presents the details for different scenarios.

V. Evaluation Setup

To demonstrate the effectiveness and potential applications of Time-Print, we build a testbed to extract fingerprints from 43 USB mass storage devices. In this section, we describe the equipment utilized, the detailed data collection methodology, the read sequence utilized, and how we denote the training and testing datasets.

A. Experimental Devices

We utilize the following devices and system configurations to gather fingerprints.

Host System, OS, and Driver Modifications. Our host system is a DELL T3500 Precision tower. The system contains an Intel Xeon E5507 4 core processor with a clock speed of 2.27GHz and 4GB of RAM. The USB 2.0 controllers are Intel 82801JI devices. We utilize a Renesas uPD720201 USB 3.0 controller (connected via PCI) for USB 3.0 experiments.

The host runs Ubuntu 18.04 LTS and we modify the USB storage drivers as detailed in Section IV-A[2]. Namely, we modify the USB driver to record the timing information for the start and completion of each USB packet transmission that is a part of the USB storage stack. Each time a device is connected, a data structure is created to store the timestamp and packet metadata information. This data structure is deleted upon device disconnect. A character device is inserted into the USB driver code to facilitate the transfer of this timing information to log files after the completion of drive fingerprinting operations.

USB Devices. We test the performance and applicability of Time-Print with 12 unique USB models and 43 different USB devices. Table I lists the device manufacturer, name, size, controller, number of devices, and protocol for every device used in our experiments. We select these brands to create a broad dataset that contains a number of the most popular devices on the market (purchased by users on Amazon as of September 2020). Each device is analyzed with no modifications to the firmware of the device. To ensure fairness, all devices are zeroed and formatted as FAT32 with an allocation size of 4KB, and are identically named as 'USB_0'. We extract the device controller name by using Flash Drive Information Extractor [52]. Of note, SanDisk does not publicly identify the versions of their flash controllers and simply reports the name 'SanDisk'.

USB Hub and Ports. To facilitate testing of the USB drives, we utilize an Amazon Basics USB-A 3.1 10-Port Hub that we connect to the inbuilt USB 2.0 Intel 82801JI hub on the host for USB 2.0 experiments and to the Renesas uPD720201 USB 3.0 hub for USB 3.0 testing.

B. Data Collection

Given our setup, we implement a script to gather data from multiple USB devices at once. The Amazon Basics USB hub utilized in our experiment can selectively enable/disable the power connected to each specific port. We implement this functionality through the `uhubctl` [63] library and simulate the physical unplugging and replugging of each USB device between every sample.

To reduce any impact on the precision of the timing within the driver, which is of the utmost importance for fingerprint, we utilize the Linux `cpuset` utility to isolate the USB storage driver process to its own CPU core. This largely prevents interference from context switches. Furthermore, since some CPUs do not guarantee that the TSC is synchronized between cores, it is necessary to ensure that all measurements are gathered from the same core.

To better explain the overall testing methodology, we further present the sample acquisition process with an example of 10 different USB drives. Before testing, each port on the USB hub is disabled such that no power is provided to a plugged-in device. We then plug each drive into a port on the USB hub and

[2]Since Time-Print is entirely software-based, it could reasonably be extended to macOS and Windows with cooperation from developers.

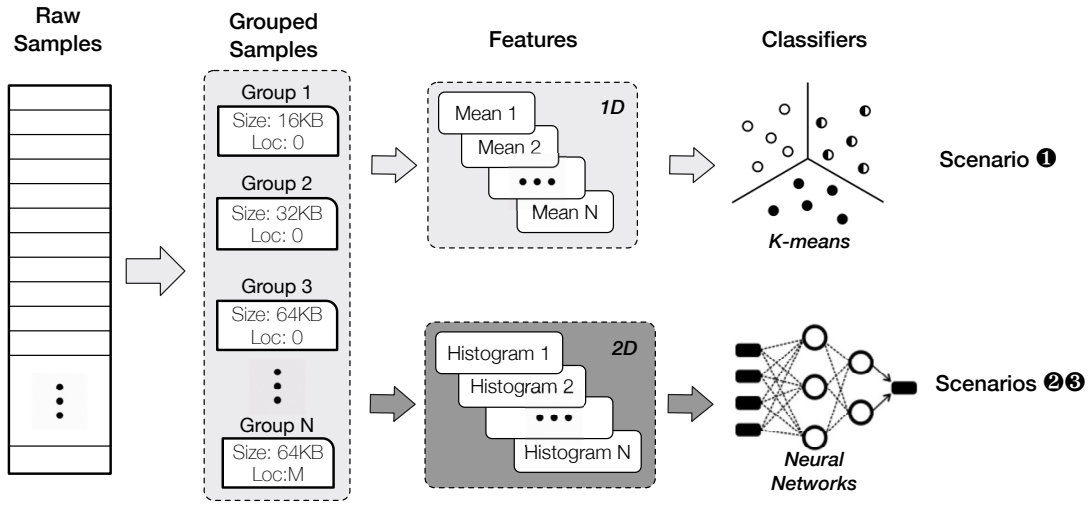

Fig. 5: Flow of generating 1D features from the raw fingerprint samples of a drive as used for different model identification (top) and 2D features as used for individual device classification (bottom).

record the mapping of the hub port to drive ID (to match each sample to a specific drive). The fingerprint gathering script enables the first port on the USB hub and waits for the USB driver process to be launched. Upon launch, the driver process is isolated to a single core of the CPU to ensure maximum timing precision. Next, we launch the fingerprinting script that initiates a series of reads of different sizes and in different locations on the drive. The returned data is not recorded because only the timing information of these reads is important. Once the collection script completes, we mount the character device and write all of the recorded timing information to a log file. The system then unmounts the character device and USB device and disables the USB port to simulate unplugging the device. We also simulate non-idle system states: the Linux `stress` utility is run to fully utilize one CPU core on every other sample. The above process is repeated for the next port on the USB hub. All drives are tested in a round-robin fashion.

Once 20 fingerprints have been gathered from each drive, we physically unplug each drive and plug it into a different port on the USB hub; this ensures that any difference observed in the readings is caused by the individual USB drives, not the USB port.

C. Fingerprint Script

To gather a fingerprint, we utilize a script of 2,900 reads. Each read is randomly chosen to be of size 16KB, 32KB, or 64KB, and to access six logical blocks at 0x0, 0x140000, 0x280000, 0x3c0000, 0x500000, 0x640000. The block addresses are spread out in an attempt to access diverse physical locations on the drive. To ensure that any uniqueness observed in the fingerprint is caused by physical variations in the drive accesses and not script variations, the script is randomly generated once and then used for each device.

D. Training and Testing Datasets

As mentioned above, in our experiments, fingerprints are gathered in a round-robin fashion from devices in a set of 20. After collecting 20 fingerprints for all drives, all devices are physically unplugged and then plugged into different ports. We thus refer to a group of 20 fingerprints as a 'session' of data. For all devices listed in Table I, we gather 4 sessions of fingerprints (i.e., 80 fingerprints). We then conduct 4-fold cross-validation by selecting 3 sessions for training, and 1 session for testing.

VI. Time-Print Results

To evaluate the effectiveness of Time-Print, we conduct a series of experiments in the three scenarios listed in Section II, namely, identifying devices with different brands, identifying unseen devices of the same brand, and auditing (i.e., classification on all insider devices).

A. Scenario ❶: Brand Identification

We first examine the accuracy for identifying a random (*unknown*) USB device of a brand different from approved devices. For instance, a system administrator would like to prevent a dropped device attack where a careless employee plugs in a malicious unauthorized device. While Figure 2 (in Section 2) shows that this timing-based fingerprint has the potential to be very effective, here we quantitatively evaluate all devices listed in Table I.

Approach. To accomplish this task, we expect that Time-Print trained on a specific device should always accept that device while rejecting all other devices with different models and brands. Thus, we design a single-class classification system using the K-Means algorithm. The one class classification system creates clusters of samples from the approved device and draws a decision boundary to reject any readings from devices of other brands or models.

Particularly, K-Means requires that each data sample is presented as a 1D feature list. K-Means utilizes this feature list and a distance metric to calculate a sample to sample distance by examining the features of each sample, and groups the samples into clusters. Once the algorithm converges, we

		Training Devices			
		Generic	SanDisk Cruzer Blade	Samsung Bar Plus	SanDisk Ultra
Testing Devices	Generic	**99.9%**	0.0	0.0	0.0
	SanDisk Cruzer Blade	0.0	**98.8%**	0.0	0.0
	Samsung Bar Plus	0.0	0.0	**99.7%**	0.0
	SanDisk Ultra	0.0	0.0	0.0	**99.9%**
	Other USB2	0.0	0.0	0.0	0.0
	Other USB3	0.0	0.0	0.0	0.0

TABLE II: Percentage of samples accepted when trained for each device model.

calculate the distance of each training sample to its closest cluster. The maximum distance value is then used to set a decision boundary. In our case, for a fingerprint to be accepted by the clustering algorithm, it must be within the decision boundary of one of the pre-trained clusters. We first preprocess each sample into different chunks by separating each reading based on the size and location offset of the measurement. With the size and locations grouped, we calculate the mean of each group, generating a 1D feature list for each sample, as illustrated in the upper part of Figure 5.

Training and testing. We train our one-class classifier on four types of devices: (1) the Generic Drives (10 devices), Samsung Bar Plus (4 devices), SanDisk Ultra (10 devices), and SanDisk Cruzer Blade devices (10 devices). We then test the classifier against all other devices listed in Table I. For clarity of presenting the results, we group all extra devices with the USB 3.X protocol into a set called 'other USB3', and all extra devices with the USB 2.0 protocol into a set called 'other USB2'.

For example, to test the accuracy for the Generic Drives, we have four sessions (80 fingerprints in total) of data for all ten devices in this model. For Generic Drive #1, we train the classifier using three sessions of data and test the classifier using the remaining one session of data, and the data from all other devices from different brands/models. We repeat the experiment for each Generic device and report the average accuracy.

Accuracy. The results are presented in Table II, showing very high accuracy: an average true accept rate of 99.5% while rejecting all drives of different models and brands (i.e., zero false accept rate). As mentioned in the threat model, Time-Print is mainly designed for use in a high-security system. Such a system should always reject unknown models to minimize security risks. While the true accept rate of 99.5% may still reject a legitimate device, with a very small chance, for the first trial, the user can simply re-plugin the USB drive and re-authenticate with the system. The probability of being rejected twice in a row is only 0.0025%. In other words, the probability of a legitimate device being accepted after two trials is 99.9975%, which is very close to one.

Overall, these results show that Time-Print can accurately distinguish unknown devices with different brands and models from legitimate devices.

B. Scenario ❷: Same Brand Device Identification

The second scenario requires Time-Print to identify *unseen* devices of the same brand and model, which is a much more difficult task as all devices share the same design.

Approach. To this end, we utilize a 2-D convolutional neural network for the classification task. As our task is not to locate the best possible network for classification but to demonstrate that fingerprinting a USB mass storage device is possible, we adopt a standard classification network design. For reference, our network architecture is provided in Table VI in the Appendix.

For preprocessing, similarly to Scenario ❶, we separate the raw timing information by size and location. As the script contains six possible locations and three possible sizes, the separation procedure produces 18 distinct collections of timing data for each fingerprint gathered.

To utilize these values within a neural network, we transform their raw format (a collection of numbers ranging from one to ten million) to a value range that works for neural networks (e.g. 0 to 1). Especially, we convert the data from each group to a histogram, with all data being scaled by the group global minimum and maximum values, from the entire training set. Such a method creates a fine-grained representation of the signal. This also makes sense as large reads take much longer to complete than short reads, and a full ranged histogram would contain a large amount of unimportant zero values. To ensure experimental integrity, the individual minimum and maximum ranges are recorded and used to process the testing set.

Each histogram can be represented as a 1D vector of measurement frequency, and the histograms for all groups can be concatenated together to create a 2D input vector to the classification network. This process is illustrated in the lower part of Figure 5. Another advantage of the histogram and neural network combination is that the network can rapidly be tuned to work for different drives, since the number of histogram bins, readings per size and location, or input trace can easily be adjusted while maintaining a consistent preprocessing pipeline.

Training and testing. To achieve accurate identification, system administrators can purchase multiple devices from the same brand and model to serve as 'malicious' devices to train the classifier. We emulate this scenario by examining the SanDisk Cruzer Blade, SanDisk Ultra, and the Generic drives. We have 10 devices for each model. Among the 10 devices, for training, one device is selected as the 'legitimate' drive, and 8 of the remaining 9 devices are chosen as 'malicious' drives; then the last is used as the 'unseen'[3] device for testing purposes. During training, we use 60 samples of each drive involved. During testing, we utilize the remaining 20 samples of each 'legitimate' drive and 20 samples of each 'unseen' drive.

[3]The 'unseen' device is equivalent to an attacker's 'malicious' device, and we use a different term to differentiate the malicious device in testing from those used in training.

	Generic		SanDisk Cruzer Blade		SanDisk Ultra	
	TAR	TRR	TAR	TRR	TAR	TRR
Raw	92.2%	93.8%	96.5%	89.2%	97.6%	90.6%
Augment	97.3%	91.7%	98.0%	93.5%	98.7%	91.4%

TABLE III: Average True Accept Rate (TAR) and True Reject Rate (TRR) for same model device identification.

Device Name (# of Devices)	Classification Accuracy
SanDisk Cruzer Blade (10)	98.6%
Generic Drive (10)	99.1%
SanDisk Ultra (10)	98.7%
Samsung Bar Plus (4)	98.4%

TABLE IV: Classification accuracy for each drive type in Scenario ❸.

To ensure fairness and remove any influence of randomness, we test all 90 possible combinations (10 possible 'legitimate' drives $\times$ 9 possible 'unseen' drives) and cross-validate each by rotating the samples utilized for training and testing.

Accuracy. Table III presents the results, showing a compelling average true accept rate (TAR) of 95.4% and an average true reject rate (TRR) of 91.2%.

After investigating the false acceptances, we find that most false acceptances occur in pairs. We realize that the problem of classifying an unknown drive is likely to benefit from synthetic data. Augmenting the training set with random variations (in an attempt to simulate more unknown devices), or with samples from more 'malicious' devices may better solidify the decision boundary of the network, leading to higher overall accuracy. We also augment the samples of the 'legitimate' drives, albeit with much smaller perturbations, to increase the true accept rate. We randomly select samples from the training set and perturb them with noise. This augmentation procedure improves the results, increasing the overall average accuracy to 95%. More specifically, the average true accept rate increases to 98.0%, and the average true reject rate increases to 92.2%.

Overall, these results indicate that our approach has enough information to uniquely fingerprint USB drives and that Time-Print can even detect *unseen* devices of the exact same brand and model.

C. Scenario ❸: Auditing / Classification

We finally evaluate the effectiveness of Time-Print on the auditing scenario, in which a system administrator needs to determine exactly which device had files copied to/from it (to track/identify an insider threat). We evaluate the accuracy for Time-Print to uniquely identify a single device from a pool of devices that are authorized for use.

We employ a network with a similar architecture to the one employed in Scenario ❷ and shown in Table VI of the Appendix. Since the goal is to identify each individual drive, we modify the final output layer of the network to contain the same number of neurons as devices that we attempt to classify. We utilize the same histogram transformation from Scenario ❷, where each sample is separated by size and location and then converted to a histogram for utilization in the neural network.

Similarly to Scenario ❷, we train and test (with cross-validation) a classifier for each model (i.e., only drives in one model are trained and tested), as we expect that an organization that adopts a device authentication system like Time-Print will limit the usage of USB drives to a particular model. Our classification results are listed in Table IV for the SanDisk Cruzer Blade, Generic, SanDisk Ultra, and Samsung Bar Plus devices. We can see that Time-Print achieves accuracy above 98.4% for varied devices, including those from some of the best selling manufacturers (SanDisk and Samsung). Furthermore, the data for SanDisk and the Generic devices demonstrates that the variability between drives is rich enough to create distinct classification boundaries among different drives. Finally, this data shows that USB fingerprinting is not limited to a single manufacturer or USB protocol. In short, Time-Print is able to fingerprint a USB drive within the same brand and model for accurate classification.

VII. Practicality of Time-Print

With the viability of fingerprinting USB mass storage devices demonstrated, we further examine the practicality of Time-Print in multiple aspects, including the latency of fingerprint acquisition, the impact of host system hardware variations on fingerprint accuracy, device usage, location accesses, whether just the flash controller itself can be utilized for fingerprinting, and how Time-Print might be deployed in the real world.

A. System Latency

The time to acquire the USB fingerprint varies depending upon the number of reads and the protocol used by the device (e.g., USB 2.0 or 3.0). We measure the time required to capture the fingerprint from a SanDisk Cruzer Blade USB 2.0 device and a SanDisk Ultra USB 3.0 device. The time cost of achieving the results in Section VI is an average of 11 seconds on the USB 2.0 drive and 6 seconds on a USB 3.0 drive, respectively. The time difference is expected as the components of the USB 3.0 drives are faster to support the enhanced speed of the protocol.

On the other hand, intuitively, fewer extra reads in the driver should save time, but degrade the identification accuracy. We further evaluate how the number of observed reads affects the accuracy of Time-Print by truncating the gathered samples and examining the accuracy in Scenario ❸ with the SanDisk Blade and Ultra devices. The results are presented in Figures 6 and 7. Both figures show that the accuracy decreases by at least a full percentage point when the number of samples is halved. The degradation continues gradually on the USB 2.0 device (down to 95% accuracy when 30x fewer samples are taken) and more steeply on the USB 3.0 device (reducing to 90% accuracy when 30x fewer samples are taken). Overall, even with 10x fewer samples being used, Time-Print can still achieve more than 94.5% accuracy while reducing the latency to only about 1 second, since the time required to acquire a fingerprint scales linearly with the number of extra reads.

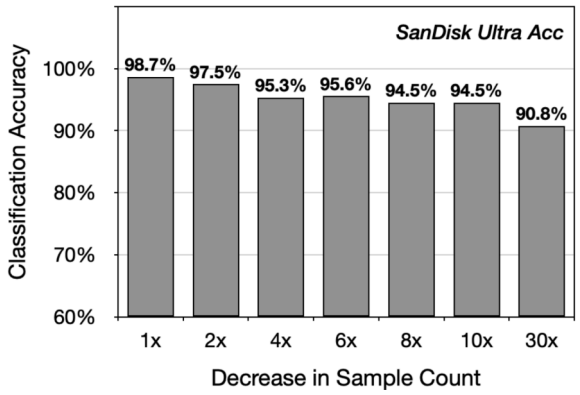

Fig. 6: Classification accuracy degradation as the number of samples is reduced (10 SanDisk Ultra USB 3.0 drives).

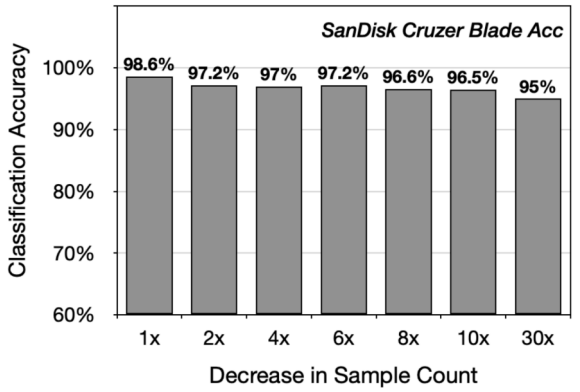

Fig. 7: Classification accuracy degradation as the number of samples is reduced (10 SanDisk Blade USB 2.0 drives).

Such a result indicates that there exists a trade-off between the time required to generate a fingerprint and the ability to use the fingerprint for unique device authentication. System administrators can utilize this knowledge to choose between the time required to obtain a fingerprint for their system and the desired security level.

B. Fingerprints with Hardware Variation

When the fingerprint data is acquired, it must pass through a myriad of system components. For example, the data transmission, beginning with the USB drive, must go through the ports and hubs along its path, through the USB controller on the motherboard, and finally through the bridge between the motherboard USB controller and the processor. Each of these system components may contain different levels of routing logic and create timing variations in the fingerprint. Therefore, we conduct several experiments to understand the impact of hardware variations on fingerprint accuracy.

Different Ports and Hubs.

To understand the impact of using different ports and hubs, we utilize the training data from Section VI, but gather new testing sets with both Generic and SanDisk Blade devices. We conduct two tests: (1) the USB hub is plugged into a different host port and (2) another Amazon Basics USB-A 3.1 10-Port Hub is used to test the accuracy of these configurations with the classifier and training data of Scenario ❸. We observe that utilizing a different host port or a different hub slightly reduces the accuracy from 99% to about 95% for the Generic devices but has no effect on the SanDisk Blade devices.

	Training System	Testing System
Processor	Intel Xeon E5507 4C/4T @ 2.27 GHz	Intel Xeon W3550 4C/8T @ 3.06 GHz
Motherboard	Dell 09KPNV	Dell 0XPDFK
RAM	2x2GB	1x8GB
USB Controller	Intel 82801JI	Intel 82801JI

TABLE V: System configurations for cross host investigation.

Different Host. We further investigate the impact of different host machines: can the same fingerprint be transferred between different host machines? We expect to see a degradation in accuracy as many factors (e.g., variations in the clock speed of the processor, motherboard, etc.) are likely to alter the fingerprint. To assess the impact, we gather a dataset on a second host system with a different configuration (system comparison is listed in Table V) using both the Generic and SanDisk Blade devices. Again, we utilize Scenario ❸ as an example to measure the accuracy degradation.

The main difference between the two host systems lies in the different CPUs. The TSC tick rate (i.e., the rate at which the TSC increments) is directly dependent on the base clock speed of CPU. Thus, we prescale the data gathered on the testing machine by multiplying the timing values by a factor of 0.7386, which is the ratio of 2.26 GHz on our training machine to 3.06 GHz on the testing machine.

With this preprocessing step, the SanDisk Blade devices experience no accuracy degradation, and the Generic drives experience an 11% accuracy decrease to 88%, which is still a promising finding. To understand the reason for these different behaviors, we uncover that the Generic devices appear to produce noisier distributions with more similar peak locations than the SanDisk Blade devices, as shown in Figure 2. We infer that such increased noise coupled with different electrical paths (e.g., different hubs, ports, machines) makes the Generic devices harder to classify in a cross host scenario. However, it should be noted that in an enterprise environment, people usually purchase a number of identical host machines with the same model of processor, motherboard, USB controllers, etc. As a result, we might experience even better fingerprint transfer between hosts. Meanwhile, this host transfer is not required in our threat model, as system administrators can train an authentication system for each protected computer.

C. Fingerprint Robustness with Device Usage

Flash devices utilize a logical to physical mapping within the flash translation layer to ensure that the flash blocks are evenly used within a device (a process called wear-leveling). When the `usb-storage` driver attempts to write data to an address, it specifies a *logical* address which the flash translation

layer converts to a physical address. Because flash blocks are modified at the block level, instead of the bit level, a write operation requires the data to be written to a new empty block and the logical to physical address mapping is updated. Since Time-Print utilizes the physical timing characteristics of specific physical blocks (accessed via logical addresses), this remapping might degrade the accuracy of the fingerprint as the device is written to.

To investigate the impact of this remapping, we conduct an experiment by writing hundreds of random files to five SanDisk Cruzer Blade devices and track the accuracy of the classification system by gathering a sample between each write. In total, we write 6,520MB of data to each 8GB drive.

The results demonstrate that Time-Print is somewhat resilient to drive writes, experiencing no accuracy degradation until about 2.3GB at which point the accuracy rapidly decreases. To better understand the cause of this sudden accuracy degradation, we examine the behavior of the actual flash drive. We utilize the tool `hdparm` to observe the actual logical block address (LBA) of each file, and notice that the drive attempts to write files to the lowest available LBA. The classification neural network essentially performs a matching task, attempting to classify the trace as the class that is the *closest* to the training samples. After more than half of the LBAs utilized for the fingerprint are written, the neural network is no longer able to perform this task reliably, since the majority of the LBAs are no longer the same. To address this problem, there are two solutions: LBA reservation and manufacturer support.

LBA reservation. If Time-Print can prevent the drive from updating the virtual to physical mapping of the blocks utilized for fingerprinting, it can prevent drive writes from affecting the fingerprint, as the drive will not reassign pages that are in use. This can be accomplished by placing small placeholder files[4] at their locations for LBA reservation. We implement this mechanism by copying large files (to occupy large swaths of LBAs) and small files into the chosen fingerprint locations, and then deleting the large files. We use the *hdparm* tool to check the LBAs used by the small fingerprint files. All of the small files combined together are only 768KB in total, thus inducing low overhead. We then write 7.3GB (the capacity of the drive) data to the drive in 16MB chunks, and observe no changes in the histograms and no accuracy degradation. This solution can adequately accommodate the normal drive usage as long as the small fingerprint files are not deleted (by users).

Manufacturer Support. This is the most resilient solution but requires collaboration with drive manufacturers. Manufacturers already provide extra flash blocks that are hidden from users to facilitate better wear leveling and drive performance. They can similarly reserve extra blocks for fingerprinting on new devices. This solution can ensure that Time-Print fingerprints are unaffected by write operations and further ensure that accidental deletion of the contents of the drive will not interfere with the fingerprint.

[4]A placeholder file should be a multiple of the remappable block size of the device, to ensure that only the placeholder file fully occupies a specific location, preventing unintentional remapping.

D. Spoofing A Fingerprint

An advanced attacker might design a malicious device to deceive Time-Print by mimicking a legitimate drive (e.g., replicate the physical fingerprint with an FPGA based system). While all of the experiments in this study utilize a static read sequence of 2,900 reads, in a real deployment, the read-sequence, including the specific locations and number of reads, can be either a secret (stored on the protected system) or randomly generated based upon a device identifier (e.g., use the serial number as a random seed). Since attackers are unable to know the exact locations utilized by Time-Print, they can only fingerprint random locations and hope that Time-Print would accept the spoofed values.

To assess the security of Time-Print against this type of advanced attack, we run an experiment where we generate random choices of locations to test whether Time-Print accepts a legitimate drive fingerprinted in the wrong locations. To emulate an attacker who is unaware of the correct sample locations, we gather a new dataset for the drives that are sampled in the wrong locations. More specifically, we generate a script that randomly chooses 6 locations on a drive and generates reads every time the drive is plugged in. We test Time-Print similarly to Scenario ❷ wherein we train Time-Print to accept samples in the correct locations of the legitimate drive and to reject samples from other devices. To further augment the training set, we add random noise to some of the training samples from the legitimate drive (similarly to Scenario ❷). Our testing set consists of the samples from the legitimate device taken in the correct locations, which should be accepted, and the samples from the legitimate device taken in the wrong locations (to emulate a spoofing attack) that should be rejected. We test this setup with the SanDisk Blade, Ultra, and Generic devices and observe an average of 96.4% true accept rate and 99.6% true reject rate. This result indicates that Time-Print is very robust against such 'spoofing' attacks.

E. Other Considerations

We further investigate whether better accuracy could be obtained by increasing the number of addresses accessed by Time-Print. Theoretically, accessing more locations on the drive should provide more information to better identify drives. To this end, we conduct experiments on accessing 18 locations (as opposed to 6), while maintaining the same number of total extra reads. We gather data on the SanDisk Cruzer Blade, Generic, and SanDisk Ultra drives, and evaluate the performance in Scenarios ❷ and ❸. We observe that while the individual accuracy of each drive type fluctuates slightly, the average performance (across all three models) in each scenario remains similar.

Another consideration is the modification of the access order. We run an experiment with five SanDisk Cruzer Blade devices by randomizing the access order for each sample. There is no accuracy degradation. We also examine whether the device format affects Time-Print. We reformat all of the devices to EXT2 and retrain Time-Print. Similarly, no accuracy degradation is observed. This is expected as the file system

format is another virtual layer above the physical pages of the USB device and therefore should not affect the fingerprint.

F. Fingerprint the Flash Controller

We also examine whether the timing information from only the flash controller could be utilized to identify the drives. We investigate this by utilizing the timing information of the 'transfer status' packet (a packet that comes only from the USB controller on the mass storage device), instead of the timing information for returning the data. We test this on both the SanDisk Cruzer Blade and the SanDisk Ultra devices. We find that utilizing only this information reduces accuracy from greater than 98% to 65% and 45% for the two types of devices, respectively. This shows that while the timing information of the flash controller can be utilized to identify some devices, it alone is insufficient to create a robust fingerprint.

G. Real-World Deployment of Time-Print

We have demonstrated that Time-Print can be utilized in various scenarios for USB drive authentication. Each of the scenarios can serve as a module in a more complete security system that might be deployed in the real world. For example, a system administrator concerned mainly about protecting systems from stray external devices can employ our system as demonstrated in Scenario ❶, while an administrator with concerns about targeted attacks might choose to utilize Scenarios ❶ and ❷ together, first rejecting unknown models and then ensuring that the device is legitimate. Scenario ❸ can be further employed to track user activities for auditing purposes. Time-Print can also be integrated into other USB security systems, which offer firewall like protections [4], [58], [59] but rely on the drive to correctly report its identification. The identification capability of Time-Print will provide a stronger defense against skilled attackers who can alter device identifiers [23], [30].

H. Future Work

Our study has demonstrated that Time-Print can accurately authenticate USB drivers from the same brand and model. In the future work, we plan to further explore the timing channel by (1) examining devices from more different scenarios, such as the same brand/model but with different capacities, (2) considering the wide deployment of Time-Print and user enrollment in practice, and (3) investigating the potential attacks against Time-Print.

In particular, strong FPGA attackers who can replay the timing information of the whole USB driver (e.g., physically unclonable function (PUF) [48] related profiling/modeling attacks) might potentially break Time-Print. Such attackers can record and profile the timing of all locations on the drive, and then answer arbitrary queries with an FPGA. However, this requires significant efforts (both time and storage) from attackers to fully understand the patterns (e.g., building the histogram for each location). In the future work, we will assess the robustness of Time-Print against such strong FPGA attacks.

VIII. Related Work

In this section, we survey the research efforts that inspired our work as well as highlight the key differences between our work and previous research.

A. Device Fingerprinting

Uniquely identifying individual physical devices has long been of interest to the security community [10], [11], [35], [46], [68]. The ability to track and authenticate a physical device accurately can help increase security and serve as another factor in multi-factor authentication. As such, many different methods for device fingerprinting have been presented.

One of the most common methods for fingerprinting is the utilization of (un)intentional electromagnetic frequency radiation. Cobb *et al.* [15], [16] showed that the process variations in the manufacturing process cause subtle variations in the unintentional electromagnetic emissions, which can be utilized to generate a valid fingerprint for similar embedded devices. Cheng *et al.* [14] further found that unique fingerprints can be created for more sophisticated systems like smartphones and laptops. Other prior works [9], [20], [43], [47] study the fingerprint generation in radiating electromagnetic signals for communication (e.g. Zigbee, WiFi, etc.). The most similar work to Time-Print is Magneto [27], which uses the unintentional electromagnetic emissions during device enumeration on a host to fingerprint USB mass storage devices. While their work demonstrates the ability to classify different brands and models accurately, the system requires expensive measurement equipment. By contrast, our work requires no special equipment and uncovers a novel timing channel that can be used to further identify devices within the same brand and model.

Device serial numbers, descriptors, and passwords are also used to thwart the connection of unauthorized USB devices [1], [28]. These defenses inherently trust that the USB device is accurately reporting software values. TMSUI [66], DeviceVeil [55], and WooKey [7] use specialized hardware to uniquely identify individual USB mass storage devices, and as a result, most of these systems are not compatible with legacy devices. Instead, Time-Print is completely software-based and does not require any extra or specialized hardware. The USB 3.0 Promoter Group has proposed a USB 3.0 Type-C PKI-based authentication scheme [62] to identify genuine products, but these mechanisms are not designed to uniquely identify individual devices. Other prior works utilize a USB protocol analyzer [37] or smart devices [5] to identify a host system and its specific operating system by inspecting the order of enumeration requests and timing between packets [18]. Unlike those works, the objective of Time-Print is to identify the peripheral device, instead of the host.

B. Flash Based Fingerprints

Several prior works have investigated whether the properties of flash devices can be utilized for fingerprinting. For example, device fingerprints are constructed using programming time and threshold voltage variations [45], [64]. Others [22], [31], [34], [41], [50], [54], [65] further investigate the design of

physically unclonable functions in flash chips and explore the impact of write disturbances, write voltage threshold variation, erase variations, and read voltage threshold variation. Sakib *et al.* [49] designed a watermark into flash devices by program-erase stressing certain parts of a device.

The above techniques work at a physical level, which requires control and functionalities that may not be available in a cost-constrained, mass-market device like a USB flash drive. Time-Print only utilizes read operations (a common function available on all USB flash drives) and thus is non-intrusive. In addition, while these technologies could be incorporated into new devices, Time-Print is fully compatible with existing devices and only requires a slight modification to the host driver.

C. USB Attacks and Defenses

USB is an easy to use and trusting protocol, which immediately begins to communicate with and set up devices when they are plugged in. Tian *et al.* [60] surveyed the landscape of USB threats and defenses from USB 1.0 to USB C, showing that most existing defenses that require extra hardware do not adequately work with legacy devices. Several attacks [30], [23] have demonstrated that modifying the firmware of USB devices can rapidly subvert a system.

Many defenses have been proposed to mitigate the problem. For example, the TPM (trusted platform module) has been used to protect sensitive information [6], [12]. GoodUSB [58] attempts to thwart firmware modification attacks [30] by creating a permission system so that users can specify permissions for devices. VIPER [38] proposes a method to verify peripheral firmware and detect proxy attacks via latency based attestation. Hernandez *et al.* [24] automatically scanned USB firmware for malicious behaviors. USBFILTER [59] presents a firewall in the USB driver stack to drop/allow USB packets based on a set of rules. Similarly, Cinch [4] creates a virtual machine layer between USB devices and the host machine to act as a firewall. Johnson *et al.* [32] designed a packet parser to protect the system from malformed USB packets. Tian *et al.* [57] proposed a unified framework to protect against malicious peripherals. Other prior works like USBeSafe [33] and USBlock [40] utilize machine learning algorithms to analyze the characteristics of USB packet traffic to prevent keyboard mimicry attacks [30]. Like those works, Time-Print is a software-based approach to enhancing USB security.

IX. Conclusion

This paper presents Time-Print, a novel timing-based fingerprinting mechanism for identifying USB mass storage devices. Time-Print creates device fingerprints by leveraging the distinctive timing differences of read operations on different devices. We develop the prototype of Time-Print as a completely software-based solution, which requires no extra hardware and thus is compatible with all current USB mass storage devices. To assess the potential security benefits of Time-Print, we present a comprehensive evaluation of over 40 USB drives in three different security scenarios, demonstrating Time-Print's ability to (1) identify known/unknown device models with greater than 99.5% accuracy, (2) identify seen/unseen devices within the same model with 95% accuracy, and (3) individually classify devices within the same model with 98.7% accuracy. We further examine the practicality of Time-Print, showing that Time-Print can retain high accuracy under different circumstances while incurring low system latency.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful and constructive comments, which helped us to improve the quality of this paper. This work was supported in part by the National Science Foundation (NSF) grants DGE-1821744 and CNS-2054657 and the Office of Navy Research (ONR) grant N00014-20-1-2153.

References

[1] USBGuard. https://github.com/USBGuard/usbguard.

[2] Exploring Stuxnet's PLC Infection Process. https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=ad4b3d10-b808-414c-b4c3-ae4a2ed85560&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments, 2010.

[3] USB Implementers Forum Revision 2.0. Universal Serial Bus Power Deliver Specification, 2016.

[4] Sebastian Angel, Riad S. Wahby, Max Howald, Joshua B. Leners, Michael Spilo, Zhen Sun, Andrew J. Blumberg, and Michael Walfish. Defending against Malicious Peripherals with Cinch. In *USENIX Security Symposium*, 2016.

[5] Adam Bates, Ryan Leonard, Hannah Pruse, Daniel Lowd, and Kevin Butler. Leveraging USB to Establish Host Identity Using Commodity Devices. In *ISOC Network and Distributed System Symposium (NDSS)*, 2014.

[6] Adam Bates, Dave (Jing) Tian, Kevin R.B. Butler, and Thomas Moyer. Trustworthy Whole-System Provenance for the Linux Kernel. In *USENIX Security Symposium*, 2015.

[7] Ryad Benadjila, Arnauld Michelizza, Mathieu Renard, Philippe Thierry, and Philippe Trebuchet. WooKey: Designing a Trusted and Efficient USB Device. In *ACM Computer Security Applications Conference (ACSAC)*, 2019.

[8] Harita Bhargava and Sanjeev Sharma. Secured Use of USB over the Intranet with Anonymous Device Identification. In *IEEE Conference on Communication Systems and Network Technologies (CSNT)*, 2018.

[9] Trevor Bihl, Kenneth Bauer, and Michael Temple. Feature Selection for RF Fingerprinting With Multiple Discriminant Analysis and Using ZigBee Device Emissions. *IEEE Transactions on Information Forensics and Security*, 2016.

[10] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile Device Identification via Sensor Fingerprinting. https://arxiv.org/pdf/2002.05905.pdf, 2014.

[11] Vladimir Brik, Suman Banerjee, Marco Gruteser, and Sangho Oh. Wireless Device Identification with Radiometric Signatures. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2008.

[12] Kevin R. B. Butler, Stephen E. McLaughlin, and Patrick D. McDaniel. Kells: A Protection Framework for Portable Data. In *ACM Annual Computer Security Applications Conference (ACSAC)*, 2010.

[13] Eric Byres. The Air Gap: SCADA's Enduring Security Myth. *Communications of the ACM*, 2013.

[14] Yushi Cheng, Xiaoyu Ji, Juchuan Zhang, Wenyuan Xu, and Yi-Chao Chen. DeMiCPU: Device Fingerprinting with Magnetic Signals Radiated by CPU. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[15] William Cobb, Eric Garcia, Michael Temple, Rusty Baldwin, and Yong Kim. Physical Layer Identification of Embedded Devices Using RF-DNA Fingerprinting. In *IEEE Military Communications Conference (MILCOM)*, 2010.

[16] William Cobb, Eric Laspe, Rusty Baldwin, Michael Temple, and Yong Kim. Intrinsic Physical-Layer Authentication of Integrated Circuits. *IEEE Transactions on Information Forensics and Security*, 2012.

[17] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. Universal Serial Bus Specification, Revision 2.0, 2000.

[18] Andy Davis. Revealing Embedded Fingerprints: Deriving Intelligence from USB Stack Interactions. Technical report, nccgroup, 2013.

[19] Douglas Gilbert. sg3_utils. https://github.com/hreinecke/sg3_utils.

[20] Clay Dubendorfer, Benjamin Ramsey, and Michael Temple. An RF-DNA Verification Process for ZigBee Networks. In *IEEE Military Communications Conference (MILCOM)*, 2012.

[21] USB Implementers Forum. Defined Class Codes. https://www.usb.org/defined-class-codes.

[22] Zimu Guo, Xiaolin Xu, Mark M. Tehranipoor, and Domenic Forte. FFD: A Framework for Fake Flash Detection. In *ACM Design Automation Conference (DAC)*, 2017.

[23] hak5darren. USB Rubber Ducky. https://github.com/hak5darren/USB-Rubber-Ducky, 2016.

[24] Grant Hernandez, Farhaan Fowze, Dave (Jing) Tian, Tuba Yavuz, and Kevin R.B. Butler. FirmUSB: Vetting USB Device Firmware Using Domain Informed Symbolic Execution. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[25] Hewlett-Packard, Intel, Microsoft, NEC, ST-NXP Wireless, and Texas Instruments. Universial Serial Bus 3.0 Specification, Revision 1.0, 2008.

[26] Hewlett-Packard, Intel, Microsoft, Renesas, ST-Ericsson, and Texas Instruments. Universal Serial Bus 3.1 Specification, 2013.

[27] Omar Adel Ibrahim, Savio Sciancalepore, Gabriele Oligeri, and Roberto Di Pietro. MAGNETO: Fingerprinting USB Flash Drives via Unintentional Magnetic Emissions. *ACM Transactions on Embedded Computing Systems*, 2020.

[28] Advanced Systems International. USB-Lock-RP. https://www.usb-lock-rp.com/.

[29] Jeffrey Robert Jacobs. Measuring the Effectiveness of the USB Flash Drive as a Vector for Social Engineering Attacks on Commercial and Residential Computer Systems. Master's Thesis Embry-Riddle Aeronautical University, 2011.

[30] Karsten Nohl Jakob Lell. BadUSB - On Accessories that Turn Evil. Blackhat USA, 2014.

[31] Shijie Jia, Luning Xia, Zhan Wang, Jingqiang Lin, Guozhu Zhang, and Yafei Ji. Extracting Robust Keys from NAND Flash Physical Unclonable Functions. In *Conference on Information Security (ISC)*, 2015.

[32] Peter C. Johnson, Sergey Bratus, and Sean W. Smith. Protecting Against Malicious Bits On the Wire: Automatically Generating a USB Protocol Parser for a Production Kernel. In *ACM Annual Computer Security Applications Conference (ACSAC)*, 2017.

[33] Amin Kharraz, Brandon L. Daley, Graham Z. Baker, William Robertson, and Engin Kirda. USBESAFE: An End-Point Solution to Protect Against USB-Based Attacks. In *USENIX Research in Attacks, Intrusions and Defenses (RAID)*, 2019.

[34] Moon-Seok Kim, Dong-Il Moon, Sang-Kyung Yoo, Sang-Hang Lee, and Yang-Kyu Choi. Investigation of Physically Unclonable Functions Using Flash Memory for Integrated Circuit Authentication. *Transactions on Nanotechnology*, 2015.

[35] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2005.

[36] David Kushner. The Real Story of Stuxnet, Feb 2013.

[37] Lara Letaw, Joe Pletcher, and Kevin Butler. Host Identification via USB Fingerprinting. In *International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*, 2011.

[38] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. VIPER: Verifying the Integrity of PERipherals' Firmware. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[39] Micron. NAND Flash 101: An Introduction to NAND Flash and How to Design It In to Your Next Product, TN-29-19. Technical report, 2010.

[40] Sebastian Neuner, Artemios G. Voyiatzis, Spiros Fotopoulos, Collin Mulliner, and Edgar R. Weippl. USBlock: Blocking USB-Based Keypress Injection Attacks. In *Data and Applications Security and Privacy*. Springer International Publishing, 2018.

[41] T Nguyen, Sunghyun Park, and Donghwa Shin. Extraction of Device Fingerprints Using Built-in Erase-Suspend Operation of Flash Memory Devices. *IEEE Access*, 2020.

[42] National Institute of Standards and Technology. Security and Privacy Controls for Federal Information Systems and Organizations, 2020.

[43] J.L. Padilla, P. Padilla, J.F. Valenzuela-Valdés, J. Ramírez, and J.M. Górriz. RF Fingerprint Measurements for the Identification of Devices in Wireless Communication Networks Based on Feature Reduction and Subspace Transformation. *Measurement*, 2014.

[44] Raymond Pompon. Attacking Air-Gap-Segregated Computers. https://www.f5.com/labs/articles/cisotociso/attacking-air-gap-segregated-computers, 2018.

[45] Pravin Prabhu, Ameen Akel, Laura M. Grupp, Wing-Kei S. Yu, G. Edward Suh, Edwin Kan, and Steven Swanson. Extracting device fingerprints from flash memory by exploiting physical variations. In *Trust and Trustworthy Computing*. Springer Berlin Heidelberg, 2011.

[46] Sakthi Radhakrishnan, A. Selcuk Uluagac, and Raheem Beyah. GTID: A Technique for Physical Device and Device Type Fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2015.

[47] Benjamin Ramsey, Michael Temple, and Barry Mullins. PHY Foundation for Multi-Factor ZigBee Node Authentication. In *IEEE Global Communications Conference (GLOBECOM)*, 2012.

[48] Ulrich Ruhrmair and Jan Solter. PUF modeling attacks: An introduction and overview. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014.

[49] Sadman Sakib, Aleksandard Milenković, and Biswajit Ray. Flash Watermark: An Anticounterfeiting Technique for NAND Flash Memories. *IEEE Transactions on Electron Devices*, 2020.

[50] Sadman Sakib, Md Rahman, Aleksandar Milenković, and Biswajit Ray. Flash Memory Based Physical Unclonable Function. In *IEEE SoutheastCon*, 2019.

[51] Paul Sawers. US Govt. plant USB sticks in security study, 60% of subjects take the bait. https://thenextweb.com/insider/2011/06/28/us-govt-plant-usb-sticks-in-security-study-60-of-subjects-take-the-bait/, 2011.

[52] ANTSpec Software. Flash Drive Information Extractor, 2019.

[53] Steve Stasiukonis. Social Engineering, the USB Way. https://www.darkreading.com/attacks-breaches/social-engineering-the-usb-way/d/d-id/1128081, 2006.

[54] Soubhagya Sutar, Arnab Raha, and Vijay Raghunathan. Memory-Based Combination PUFs for Device Authentication in Embedded Systems. *Transactions on Multi-Scale Computing Systems*, 2018.

[55] Kuniyasu Suzaki, Yohei Hori, Kazukuni Kobara, and Mohammad Mannan. DeviceVeil: Robust Authentication for Individual USB Devices Using Physical Unclonable Functions. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[56] The Wireshark Team. Wireshark. https://www.wireshark.org/.

[57] Dave Tian, Grant Hernandez, Joseph Choi, Vanessa Frost, Peter Johnson, and Kevin R. B. Butler. LBM: A Security Framework for Peripherals within the Linux Kernel. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

[58] Dave Jing Tian, Adam Bates, and Kevin Butler. Defending Against Malicious USB Firmware with GoodUSB. In *ACM Annual Computer Security Applications Conference (ACSAC)*, 2015.

[59] Dave (Jing) Tian, Nolen Scaife, Adam Bates, Kevin Butler, and Patrick Traynor. Making USB Great Again with USBFILTER. In *USENIX Security Symposium*, 2016.

[60] Jing Tian, Nolen Scaife, Deepak Kumar, Michael Bailey, Adam Bates, and Kevin Butler. SoK: "Plug & Pray" Today – Understanding USB Insecurity in Versions 1 Through C. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[61] Matthew Tischer, Zakir Durumeric, Sam Foster, Sunny Duan, Alec Mori, Elie Bursztein, and Michael Bailey. Users Really Do Plug in USB Drives They Find. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[62] USB-3.0-Promoter-Group. Universial Serial Bus Type-C Authentication Specification Release 1.0 with ECN and Errata, 2017.

[63] Vadim Mikhailov. uhubctl. https://github.com/mvp/uhubctl.

[64] Yinglei Wang, Wing kei Yu, Shuo Wu, Greg Malysa, G. Edward Suh, and Edwin Kan. Flash Memory for Ubiquitous Hardware Security Functions: True Random Number Generation and Device Fingerprints. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[65] Sarah Xu, Wing kei Yu, G. Edward Suh, and Edwin Kan. Understanding Sources of Variations in Flash Memory for Physical Unclonable Functions. In *International Memory Workshop (IMW)*, 2014.

[66] Bo Yang, Yu Qin, Zhang Yingjun, Weijin Wang, and Dengguo Feng. TMSUI: A Trust Management Scheme of USB Storage Devices for Industrial Control Systems. In *Information and Communications Security*, 2016.

[67] Pete Zaitcev. The usbmon: USB Monitoring Framework, 2005.

[68] Jiexin Zhang, Alastair Beresford, and Ian Sheret. SensorID: Sensor Calibration Fingerprinting for Smartphones. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

APPENDIX A
ADDITIONAL TABLES

Layer	Type	Kernel Size	# of Filters/Neurons
1	2D Convolution	(1,3)	8
2	2D Max Pool	(1,2)	-
3	2D Convolution	(1,3)	16
4	2D Max Pool	(1,2)	-
5	2D Convolution	(1,3)	128
6	2D Max Pool	(1,2)	-
7	Flatten	-	-
8	Dropout	.1	-
9	Dense	-	50
10	Dense	-	50
11	Dense	-	2

TABLE VI: Neural network architecture used for classification.