HubBub: Contention-Based Side-Channel Attacks on USB Hubs

Junpeng Wan[1], Yanxiang Bi[2], Han Gao[1], Dave (Jing) Tian[1]

[1]*Purdue University*, [2]*The Chinese University of Hong Kong*
{wan155, coolgao, daveti}@purdue.edu, by022@ie.cuhk.edu.hk

Abstract

Universal Serial Bus (USB) hubs enhance connectivity in modern computers by allowing multiple peripheral devices to share a single upstream port. Common peripherals include external storage devices, network interface cards, cameras, and keyboards. However, when several devices operate simultaneously, bus contention within the USB hub becomes unavoidable. Such contention causes timing variations that can be exploited to leak sensitive information.

We identify three types of USB bus contention and design multiple side-channel attacks to infer user activities based on these contentions. These attacks can be launched from a virtual machine, a remote website, or a USB peripheral, as demonstrated in three distinct attack scenarios. By collecting I/O interval data using our probers, we can recover information such as web browsing history, camera-captured activities, and keystrokes with accuracies ranging from 85% to 99%. We evaluated 15 leading USB 3.x external hubs on the market, a USB 2.0 hub, and an internal hub, most of which are vulnerable to HubBub attacks. We have reported our findings to the relevant stakeholders.

1 Introduction

Universal Serial Bus (USB) is the *de facto* interface between computers and peripherals. As portable computers such as laptops and small-factor PCs become slimmer and more compact, manufacturers further reduce the number of USB ports available in their products to reduce the weights and overall sizes. In addition to internal USB hubs usually built into the motherboard, external USB hubs connecting to a computer's existing USB ports are also widespread and continue to grow in popularity. According to a report [47], the global USB hub market, valued at USD 4.4 billion in 2023, is projected to double by 2032.

A typical USB hub features one upstream port that connects to the host computer and multiple downstream ports for various peripherals, such as keyboards, Ethernet Network Interface Cards (NICs), and cameras. These peripherals frequently exchange sensitive information with the host, including private browsing data, personal activities captured by the camera, and even user-typed passwords. However, when multiple USB devices are connected to a single hub and operate simultaneously, USB bus contention becomes unavoidable. This contention can degrade performance and even introduce privacy risks.

In this work, we identify and analyze three types of USB hub contention: *Traffic Congestion*, *Scheduling Contention*, and *Shared-structure Contention*. Traffic Congestion occurs when the combined throughput of connected devices exceeds the bandwidth capacity of the USB hub, resulting in performance degradation. Scheduling Contention stems from the host-centric nature of USB communication, where additional latency is introduced as the host orchestrates data transfers among competing devices, especially those with different transfer priorities. Shared-structure Contention arises from limitations in the hub's internal architecture, such as shared components like Transaction Translators, leading to increased latency at each operation.

By exploiting USB hub contention, we introduce ***HubBub***, the first contention-based timing side-channel attacks on USB hubs, which can steal the victim's private information from a virtual machine, a remote webpage, or a USB peripheral. Specifically, we evaluate HubBub in three attack scenarios. In Attack A, we use a USB SSD to congest the USB hub and monitor the increased I/O operation time caused by the victim's web browsing. Using our classifier HubAttGRU, we can infer browsing history with a Top-1 classification accuracy of 98.84% and a Top-3 accuracy of 99.53% across the top 100 websites. In Attack B, we leverage a USB NIC to monitor the USB hub, affecting the transfers of USB camera data. We achieved 85% accuracy in classifying six basic daily activities captured by a webcam. In Attack C, we monitor the hub using a USB device that shares the hub with a keyboard. The attacker can detect keystroke patterns through timing variations of its own operations. By applying simple filters, we successfully identify keystrokes with a precision of 93.4%

and a recall of 92.09%.

Our contributions can be summarized as follows:

- We reveal the privacy risks of USB hubs caused by USB bus contention and realize them in three real-world attack scenarios, i.e., website inference, camera activities capturing, and keystroke detection.
- Leveraging USB hubs, we propose HubBub, the first timing side-channel attacks targeting USB hubs. We demonstrate HubBub's capability to infer the victim's private information under two threat models.
- We illustrate the security impact of HubBub by evaluating 15 USB 3.x hubs, a USB 2.0 hub, and an internal USB hub. Our findings reveal that most tested devices are vulnerable to HubBub attacks. We also demonstrate that our attacks are transferable between hubs and resilient to noise to a certain level.

2 Background

The USB protocol. To unify peripheral-computer interfaces like serial, parallel, and Android Debug Bridge (ADB) ports, the Universal Serial Bus (USB) standard was introduced first in 1996 [11]. Since then, USB has experienced considerable evolution. With 1.5 Mbps (Low-Speed) to 12 Mbps (Full-Speed) data transfer rates, USB 1.x (short for USB 1.0 and 1.1 [11, 13]) benefits mainly for peripherals with low transfer rates, like keyboards and mice. Released in 2000, USB 2.0 [12] has a bandwidth surged to 480 Mbps (High-Speed), accommodating the increasing data demands like CD writers, Ethernet adapters, and digital cameras.

Released in 2008, USB 3.0 [9] supports a transfer rate of up to 5 Gbps (Super-Speed Protocol). Released in 2013 and 2017, respectively, USB 3.1 [10] and USB 3.2 [28] offer a maximum of 10 Gbps (Super-Speed Plus Protocol) and 20 Gbps bandwidth (Super-Sped Plus with Dual Lane). Until now, USB 3.x is the dominant USB protocol.

USB policy and mechanism. USB is a host-centric communication protocol, i.e., only the host can initiate data transfers. In USB 1.x and 2.0 specifications, the host periodically polls devices, with polling intervals depending on the USB version and device speed. In the USB 3.x and later specifications, the host no longer uses regular polling but instead employs an asynchronous notification mechanism. This approach allows devices to alert the host when they are prepared to transmit or receive data.

A USB endpoint is a logical entity in a device that serves as the source or sink for data transfer. An endpoint is characterized by direction (IN or OUT), the maximum packet size, and the transfer types, including *Control*, *Bulk*, *Interrupt*, and *Isochronous* transfers [8]. Control transfers are utilized for command and control during device setup, whereas bulk transfers manage large data loads that are not time-sensitive

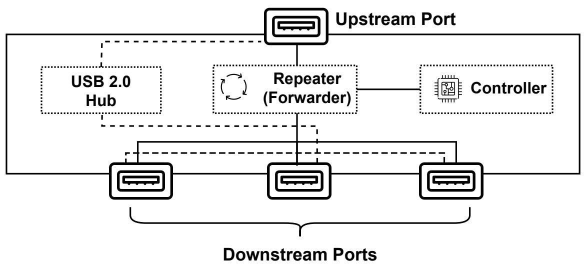


Figure 1: Topology of a USB 3.0 hub

(e.g., traffic of mass storage devices). Interrupt transfers are designed for devices that need occasional but timely data communication (e.g., keyboards and mice) with a guaranteed latency. Isochronous transfers are designed for continuous data streams (e.g., HDMI) that require consistent timing without any guarantee of error correction. USB host controllers manage bus bandwidth allocation, ensuring that no more than 90% of a full-speed frame and 80% of a high-speed microframe are allocated for *Periodic Transfers* (isochronous and interrupt). This reservation leaves at least 10% of the bandwidth for control transfers. Any remaining bandwidth is then available for bulk transfers [12]. Besides, each USB device can have up to 16 IN endpoints and 16 OUT endpoints, with endpoint 0 reserved for control transfers.

USB hubs. The function of a USB hub is to transform a single USB port into multiple ports, facilitating the simultaneous connection of peripherals. Figure 1 illustrates a typical multi-port USB hub topology. The downstream ports connect devices such as keyboards and USB flash drives, while the upstream port links to a host USB port. Traffic between the host and USB devices is handled by the Repeater/Forwarder unit under the scheduling of a micro-controller within the USB hub. To ensure backward compatibility, a USB hub includes the features of lower-version USB hubs. As demonstrated in Figure 1, a USB 3.0 (Super-Speed) hub also possesses the functionality of a USB 2.0 (High-Speed) hub. Likewise, a USB 2.0 hub contains a component known as a *Transaction Translator (TT)*, which facilitates the translation between the USB 2.0 packets (High-Speed) and the USB 1.1 devices (Low-Speed and Full-Speed).

In the hierarchy of USB hubs, the root hub refers to the hub that is directly linked to the host computer's USB controller [28]. The root hub is also an internal USB hub because it is integrated into the computer's motherboard. External USB hubs, instead, connect their upstream ports with downstream USB ports of internal USB hubs.

3 Attack Overview

In this section, we first outline the threat model of HubBub attacks, analyze USB traffic, and highlight three types of hub contention. Next, we discuss the attack procedures. Finally, we describe the challenges of designing and applying the attacks.

3.1 Threat Model

In the context of HubBub attacks, a USB hub with multiple downstream ports is connected to the host computer, as illustrated in Figure 2. The hub can be internal or external, supporting USB 2.0 or 3.x specifications. We assume that one of the target devices, such as a camera, a keyboard, or a Network Interface Card (NIC), is attached to one downstream port of the hub. The victim may use the target device in various ways, like monitoring a room with the camera, typing passwords on the keyboard, or browsing websites via the NIC. In our settings, both the USB hub and the host computer are assumed to be benign and beyond the attacker's control. However, the attacker can still monitor the victim's activities from the USB hub in the following two threat models.

In Threat Model I, we assume that the attacking program is pre-installed, e.g., via social engineering, but is isolated from other components of the host system, as illustrated in Figure 2. This threat model aligns with the previous studies assuming a virtual machine managed by a Virtual Machine Monitor (VMM) [29, 30, 68, 71, 80] or a website confined within a web browser [16, 22, 23, 32, 41]. As a result, the attacker cannot access the victim's memory space outside the confinement, preventing any tampering outside the isolation or access to sensitive information from the victim program.

Additionally, we assume the attacker has access to one of the USB peripherals connected to the USB hub, such as a USB SSD or a USB NIC. As shown in Figure 2, we design two probing tools in this threat model: the *SSD Prober*, which monitors the USB hub traffic by operating with files on the SSD, and the *NIC Prober*, which monitors the USB hub traffic by sending and receiving packets via the NIC. Both the USB SSD and NIC are common peripherals connected to a hub.

Threat Model II is consistent with existing USB attacks [5, 6, 18, 50, 66, 69], where malicious USB devices are introduced into the victim's environment through social engineering tactics. These devices may appear as enticing lost drives, legitimate peripherals like keyboards or network adapters, tampered by a supply chain attack, or embedded in compromised public charging stations. In this threat model, we assume the attack program resides in one of the USB devices connected to the hub. This device operates in a plug-and-play fashion, automatically monitoring the USB hub traffic upon connection. In addition to traffic monitoring, it performs legitimate functions, such as acting as a microphone or a flash drive, allowing it to launch the attack covertly. In this threat model, we developed the *Gadget Prober*, a USB device equipped with a traffic monitoring program, as shown in Figure 2. In the current implementation, we developed a custom USB host driver for the Gadget Prober, which lacks a default driver on the host. This driver passively receives USB packets from the gadget without performing any malicious or suspicious operations. An attacker could potentially bypass this requirement by emulating a device that uses standard host drivers, as shown in BadUSB attacks [50].

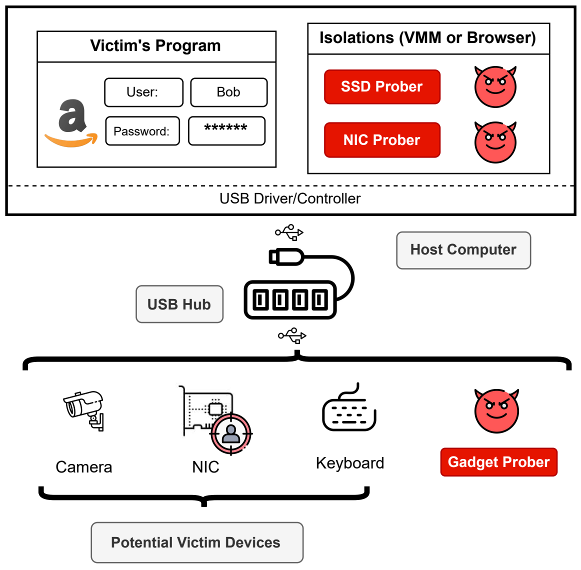


Figure 2: Overview of HubBub Attacks.

In later sections, attacks A, B, and C utilize the SSD Prober, NIC Prober, and Gadget Prober, respectively. All three probers appear benign and do not contain malicious behaviors explicitly, such as writing shellcodes to storage or downloading malicious software from external websites. All the probers operate covertly and evade the detection of conventional security measures, such as antivirus software. In both threat models, alternative attacks that rely on system configurations other than USB hub sharing are also possible. We discuss their implications and limitations in Section 9 on related work.

3.2 Characterization of USB Hub Contention

In USB hubs, every downstream device shares the same hub structure and upstream port, thereby causing unavoidable contention. We classify them into the following three types.

Traffic Congestion. Traffic Congestion occurs when the data load exceeds the capacity of the USB hub, particularly when multiple downstream devices are in use simultaneously. Our review of popular USB hubs on the market indicates that most of them are either USB 3.0 or 3.1, with theoretical transfer speeds of up to 5 Gbps and 10 Gbps, respectively. In reality, these hubs typically achieve bandwidths of only 400 MB/s and 1,100 MB/s due to encoding overhead [10]. Meanwhile, the sequential read speed of a SAMSUNG MU-PC500T portable SSD can reach 1,050 MB/s, which alone can saturate the bandwidth of USB 3.0 hubs. When combined with other devices, it can also exceed the bandwidth of USB 3.1 hubs.

Scheduling Contention. USB transfer scheduling can introduce delays even when the bus bandwidth is not fully utilized. This is because USB is a host-centric protocol, where the host initiates and schedules data transfers for all connected devices. The host schedules these transfers with the granularity of time interval, either frame (1 ms) or microframe (125 µs). A transfer does not happen immediately, thus introducing scheduling delays, particularly for upstream traffic when multiple devices compete for bandwidth. Furthermore, the USB protocol prioritizes different types of transfers: *periodic transfers*, i.e., interrupt and isochronous transfers, are given precedence over *asynchronous transfers*, i.e., bulk and control transfers. This prioritization can further exacerbate delays for lower-priority transfers in high-traffic scenarios.

Take the High-Speed (USB 2.0) protocol as an example. Each microframe can accommodate up to 3 isochronous transfers, 3 interrupt transfers, or 13 bulk transfers. Figure 3 illustrates a sample traffic profile of USB 2.0 traffic across three microframes, each lasting 125 µs, demonstrating how different transfer types are allocated. At the start of each microframe, isochronous transfers from a camera are scheduled first to ensure consistent data delivery. Bulk transfers, such as those from NIC and storage devices, are placed later in the microframe. When the keyboard detects a key press, it sends a packet via interrupt transfer, which is scheduled after the isochronous transfers but before bulk, highlighted in red. From Figure 3, we can see that traffic using bulk transfers may experience extra scheduling delays when periodic transfers are present.

Shared-structure Contention. While USB hubs generally provide multiple downstream ports, as shown in Figure 1, certain hardware components of the USB hub cannot be shared simultaneously. For example, USB 2.0 hubs use Transaction Translator (TT) [12] to translate and buffer low-speed and full-speed USB traffic to high-speed, ensuring backward compatibility. These hubs often feature a *Single-TT* architecture shared by multiple USB 1.1 devices, leading to bus contention and performance degradation, as a TT can only process one transaction at a time. Even with a single USB 1.1 device connected, the TT still introduces a translation delay, known as *TT think time* [12]. Because TT manages internal resources, buffers data, and ensures proper synchronization, the aforementioned delay arises from interpreting the *Start-split* and *Complete-split* transactions from the USB 2.0 host and establishing the corresponding full-speed or low-speed transactions on the downstream bus.

3.3 Attack Steps of HubBub

Step 1: Identify the target USB device. Initially, the attacker needs to identify the target devices connected to the USB hub. Take Figure 2 as an example. If the victim is browsing websites such as *amazon.com*, the target device is the USB NIC; if the victim is typing a password to log in, the target device becomes the keyboard. The attacker might have prior knowledge of these devices, e.g., through social engineering techniques. Additionally, the attacker may identify the target device by observing USB traffic using our provided probers.

Step 2: Monitor hub traffic Online. Based on different targets, the attacker may choose to deploy different probers, e.g., the SSD Prober, the NIC Prober, or the Gadget Prober. The probers perform I/O operations to generate traffic passing through the USB hub and record the time interval of each operation. The USB traffic related to the victim's activities then causes disturbances in the time intervals of the attacker's operations. The attacker subsequently collects a sequence of time intervals, which captures the victim's traffic patterns indirectly.

Step 3: Infer private information offline. The attacker analyzes the time-interval sequences gathered in Step 2, where longer intervals indicate the presence of victim activities on the USB hubs. Depending on the specific scenario, the attacker applies various analytical methods to these time-series patterns to extract the victim's sensitive information. In the scenario of Figure 2, the attacker can infer the website visited (e.g., *Amazon.com*) or the password entered by the victim.

3.4 Challenges

The design and implementation of HubBub attacks presents several practical challenges. First, exploiting Traffic Congestion requires our prober to: 1) maintain high throughput to saturate the bus bandwidth and capture the extra latency caused by the victim's traffic; 2) achieve high *Input/Output Operations Per Second (IOPS)*) for fine-grained activity monitoring. Besides, we need to meet the above requirements simultaneously, which requires a suitable trade-off.

Second, different from protocols like PCIe [53], USB data transfers are strictly host-driven [13], which complicates potential attacks originating from USB devices.

Third, USB hubs vary in specification and design. An attack that works on a single USB hub does not guarantee that it will work on other hubs. Therefore, we need to evaluate different models to ensure the generality and transferability of the attack. Moreover, given the wide variety of USB peripherals, exploring multiple usage scenarios is essential to assess HubBub's impact in the real world.

4 Attack A: Website Inference

In this attack, we infer the websites visited by the victim by leveraging traffic congestion in the hub. As described in Threat Model I in Subsection 3.1, we assume that a USB hub is connected to the host computer, with both a portable SSD and a USB NIC attached. Moreover, the attacker program can be isolated within a virtual machine yet still have access to the SSD. The USB NIC serves as the means of network connection for the host and acts as the target device. Meanwhile, the

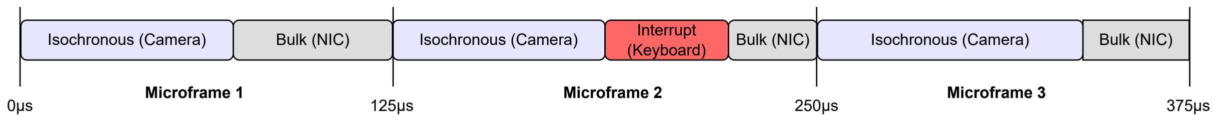


Figure 3: USB Transfer Scheduling in Three Microframes

SSD functions as the probing device, monitoring the network activities from the NIC.

4.1 SSD Prober Design and Implementation

First, we describe how to design the SSD Prober to address the first challenge mentioned in Subsection 3.4, namely, ensuring high throughput while balancing the trade-off to achieve a high IOPS.

High throughput and prober implementation. To maximize the throughput in the hub, we make several key design choices for the SSD Prober. Firstly, by leveraging the features of an SSD, we use READ operations instead of WRITE operations, as reading is faster and writing requires erasing and programming data [65]. Secondly, we perform sequential rather than random operations to make caching and prefetching of data blocks more effective. Finally, we generate I/O commands asynchronously, which allows multiple I/O operations to be initiated and processed concurrently, increasing the throughput and reducing overall wait times.

Based on the above design options, we implement the SSD Prober using *IO_uring* [7], a Linux asynchronous I/O framework, to initiate sequential READ operations. The SSD Prober consists of 166 lines of C code, simplified in Algorithm 1. As shown in the algorithm, we submit READ commands to continuously keep the Submission Queue (`SQ`) full, ensuring a steady stream of I/O operations. We then check the status of the READ operations by monitoring the completion queue (`CQ`) and record timestamps (`TS`). The program also ensures that the submission queue remains full during the loop to maintain continuous operation. Additionally, we enable `O_DIRECT` flag when we open the file to bypass the operating system's page cache and perform direct I/O operations. To further increase the throughput, we enable the kernel polling mode of `IO_uring` by setting the `IORING_SETUP_SQPOLL` flag during initialization. With this mode, a kernel thread polls the submission queue rather than invoking `io_uring_enter()` via the user program, providing a high I/O command submission speed [7]. Before Linux 5.11, root privileges were required to use this feature. Linux 5.11 introduced support for non-root users with the `CAP_SYS_NICE` capability, and this restriction was further lifted in 5.13, allowing unprivileged users without any special privileges [46]. Note that the SSD prober can be deployed both natively in a host or inside a virtual machine.

We conduct the following analysis to demonstrate that the SSD Prober is capable of saturating typical USB hubs.

Algorithm 1 SSD Prober

```
Input: io_size, ring_size, access_num
Output: Intervals
 1: Intervals ← [ ]
 2: SQ ← INITRING(ring_size)                         ▷ Init Submission Queue
 3: CQ ← INITRING(ring_size)                         ▷ Init Completion Queue
 4: SUBMIT(ring_size * READ(io_size)) → SQ
 5: TS ← GETTIME()                                   ▷ TS = timestamp
 6: while LEN(Intervals) < access_num do
 7:     FINISH(READ) → CQ
 8:     Intervals.APPEND(GETTIME() − TS)
 9:     TS ← GETTIME()
10:     current_size ← LENGTH(SQ)
11:     if CurrentSize < ring_size then
12:         read_needed = ring_size - current_size
13:         SUBMIT(read_needed * READ(io_size)) → SQ
14:     end if
15: end while
```

Firstly, we deploy the SSD Prober in the host computer to read files within SSD1[1], which is connected via a USB 3.0 Hub named A1. As depicted in Figure 4, when `IO_SIZE` is 16K, the throughput of the SSD prober is approximately 3.6 Gbps, or around 432 MB/s. According to the USB 3.0 specification, the effective bandwidth of USB 3.0 is approximately 400 MB/s [9], indicating that our prober successfully saturates the hub. Note that although the theoretical maximum throughput of USB 3.0 is 5 Gbps or 625 MB/s, the actual effective bandwidth is around 400 MB/s due to: 1) the use of 8b/10b symbol encoding, which imposes a 20% overhead [9], and 2) the overhead introduced by link-level flow control, structuring, and framing of USB packets.

Then we test the prober on a USB 3.1 hub, B1, to read files within SSD2[2]. As shown in Figure 4, when setting `IO_SIZE` to 16K, our prober achieves a throughput of approximately 8 Gbps, or 1,000 MB/s, which is close to the effective bandwidth of 1,100 MB/s for USB 3.1, as specified [10]. Thus, the SSD prober can saturate both USB 3.0 and USB 3.1 hubs.

Throughput-IOPS trade-off. As shown in Figure 4, increasing the size of each I/O operation enhances bandwidth utilization. However, if the I/O size becomes too large, the time interval of each operation also increases. As a result, the granularity of the captured patterns becomes coarse, limiting the effectiveness of inferring the information from the victim. Figure 4 illustrates the trade-off between throughput and IOPS as

[1]SSD1 is a portable USB SSD that supports throughput up to 540 MB/s.
[2]SSD2 is another USB SSD with a throughput of 1,050 MB/s.

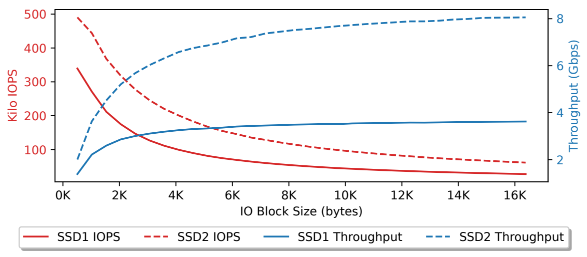


Figure 4: IOPS and Throughput of the SSD Prober as `IO_SIZE` varies from 0 to 16 K. SSD1 is connected to a USB 3.0 hub, while SSD2 is connected to a USB 3.1 hub.

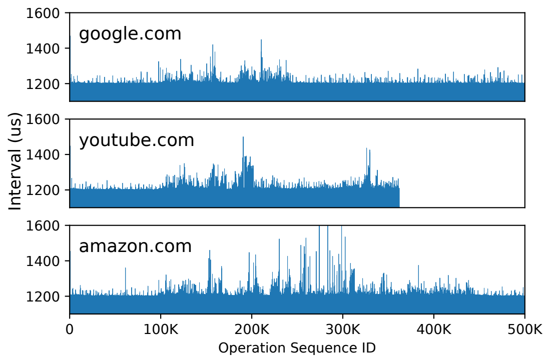


Figure 5: Website Patterns from the SSD prober

we vary `IO_SIZE` in Algorithm 1 from 512 to 16,384 (16K). Notably, as throughput increases, IOPS decreases, necessitating a careful balance between them.

In the design of the SSD prober, we prioritize throughput over IOPS initially and then balance both metrics. This is because the traffic of the SSD prober can only interact with the victim traffic when the throughput is sufficiently high for competition. Based on the above analysis and Figure 4, we set `IO_SIZE` to 4906 for USB 3.0 hubs and 8192 for USB 3.1 hubs to optimize the trade-off between throughput and IOPS. With these parameters, both USB 3.0 and USB 3.1 hubs can be saturated while maintaining a high IOPS of around 100K, with each I/O operation around 10 microseconds.

Timing function and parameters selection. To achieve high precision for recording timestamps, we use `rdtscp` [25] on the x86-64 ISA to access the processor's *timestamp counter (TSC)*. The TSC records the number of CPU clock cycles since the last reset, providing precise timing measurements.

For the remaining input parameters in Algorithm 1, we set `RING_SIZE` to 128, which offers acceptable performance as analyzed above. `ACCESS_NUM` is configured to 0x80000, determining the total collection duration, which makes the prober run for approximately 5.3 seconds on USB 3.0 hubs and around 4.6 seconds on USB 3.1 hubs.

4.2 Website Fingerprinting

In our experiment, we connected SSD1 and the Ethernet cable to a USB 3.0 hub named A2. We then deployed the SSD Prober and visited websites from the desktop, which utilizes the network connection provided by a USB NIC attached to A2. Due to variations in traffic patterns, such as loading images, documents, videos, and unique element-loading sequences, the traffic of different websites becomes distinguishable. Figure 5 showcases the `Intervals` (output of Algorithm 1) from the SSD Prober for three popular websites: *google.com*, *youtube.com*, and *amazon.com*, each exhibiting distinct patterns. Due to the discriminable patterns captured by the SSD prober, we can differentiate a number of websites using classifiers, as detailed in Subsection 4.2.3. Furthermore, we can deploy the SSD prober in a virtual machine (VM), as detailed in E3 of Subsection 7.2.

4.2.1 Dataset Creation

To evaluate the performance of our attack on website fingerprinting, we created a new dataset, the HubBub Website Fingerprinting (HF) Dataset. Our dataset is different from previous website fingerprinting datasets, such as the DF dataset [61], the Wang dataset [72], and the AWF dataset [55], which capture direct features like packet size, direction, and timestamps. In contrast, the HF dataset only comprises indirect features, i.e., I/O operation intervals recorded by our probers.

The HF Dataset comprises three subsets: *HubBub-100*, *HubBub-45*, and *HubBub-OpenWorld*. HubBub-100 is designed to evaluate the attack's performance on globally leading top websites, featuring 53,700 traces collected from the top 100 popular websites. HubBub-45 focuses on assessing the impact of HubBub across 15 USB 3.x hubs and 45 websites, with 135,000 traces captured in total. Lastly, HubBub-OpenWorld contains website fingerprints for the open-world setting, incorporating 5,636 traces from 2,818 websites. Besides, we use Chrome in *headless* mode to simulate user website visiting from the USB host. More details about HF dataset can be found in Appendix A.

4.2.2 Data Pre-processing

To reduce noise and improve classifier performance, we apply two operations to each `Interval`.

Windowed maximum. We divide the interval sequence into non-overlapping windows with a window size of 64 and extract the maximum value from each window. This parameter is selected after analyzing the traces, which typically contain 100 to 120 near-zero values followed by a distinct peak. With window size 64, we can significantly reduce the data size while keeping these peaks.

Smoothing. To capture overall traffic trends and reduce the impact of short-term noise, we apply a smoothing function with a window size of $M = 50$. The window slides over the sequence, computing the mean at each step. This operation smooths the data over 32 ms, reducing noise while preserving website-related patterns.

4.2.3 Design and Choice of Classifiers

Given that our dataset comprises time-interval sequences, we initially turn to *Recurrent Neural Network (RNN)* based methods, which are well-suited for learning from sequential data due to their ability to capture temporal dependencies. However, traditional RNNs often suffer from issues like vanishing gradients, which can hinder their ability to learn long-term dependencies. In order to avoid this, we explore the advanced variant, the *Gated Recurrent Unit (GRU)* [17], which incorporates gating mechanisms to enhance information flow control and lessen the drawbacks. In line with previous work in website fingerprinting [31, 68, 75], we also incorporate the attention mechanism into our model, which has proven highly effective by allowing the model to focus on the most relevant parts of the input sequence, thereby emphasizing critical features within the time-interval sequence. To this end, we developed *HubAttGRU* (HubBub Attention-Based GRU Classifier), which leverages an attention-based GRU architecture to infer the websites. The detailed structure is provided in Table 6 in the appendix.

Additionally, we evaluate the performance of the HubBub attack using both CNN-based and traditional machine learning classifiers. First, CNN-based models are more time-efficient compared to RNNs. We reproduce three CNN-based models: 1) *Deep Fingerprinting (DF)* [61], a multi-layer 1D-CNN model; 2) *Deep Nearest Neighbor Fingerprinting (DNNF)* [26], which uses a CNN for feature extraction followed by KNN clustering on the final convolutional layer; 3) *Triplet Fingerprinting (TF)* [62], which jointly trains three DF models to produce feature vectors that maximize intra-class similarity and minimize inter-class similarity, also using KNN for clustering.

Second, we explore classical machine learning models to validate our dataset efficiently and provide lightweight alternatives where computational resources are limited. Using *Scikit-learn* [54] version `1.0.2`, we implement five models: Support Vector Machine (SVM), Decision Tree, XGBoost, Random Forest, and Multi-layer Perceptron (MLP).

5 Attack B: Camera Activity Detection

In this attack, we assume that the attacker is located on a remote website and aims to collect information about the local environment. The website will be accessed by the victim from a browser which enforces browser isolations, such as Chrome's *strict site isolation* policy. In our attack setting, Internet access is provided via an Ethernet Adapter (or NIC) attached to the USB hub, which is connected to the USB port of a computer. A camera is also linked to the same USB hub, monitoring the room. The above assumption aligns with Threat Model I detailed in Subsection 3.1.

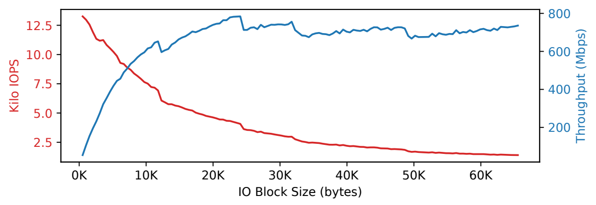


Figure 6: IOPS and Throughput of the NIC Prober as `BUFFER_SIZE` varies from 0 to 64 K.

5.1 NIC Prober Design and Implementation

As described before, the attacker will deploy the NIC Prober within the browser and receive packets from a server she controls. Thus, the NIC prober consists of a client part and a server part, simplified as Algorithm 2 and Algorithm 3 in the appendix. The client part has 130 lines of JavaScript code embedded within an HTML webpage, while another 73 lines of JavaScript code are implemented in the server part. In our implementation, we use the *WebSocket* [21] protocol to establish the connection. However, the attacker can opt for other frameworks, such as *WebRTC* [64]. For the timing function, we use `performance.now()` of JavaScript to record timestamps, which offers microsecond-level accuracy.

Throughput-IOPS trade-off. The design principle of this prober differs from that of the SSD Prober. Specifically, the NIC Prober prioritizes high IOPS over high throughput to focus on investigating scheduling contention in this attack. Since hub congestion is not required, the low latency allows the prober to capture USB host scheduling behavior with high precision.

We deploy the NIC Prober within the Chrome browser on a computer connected to USB 3.0 hub A1. This hub has a default Ethernet Adapter named ETH-D1. The server part of the prober is hosted using *NodeJS* on a desktop within the local area network. We run the NIC Prober with `BUFFER_SIZE` values ranging from 512 to 64K bytes and evaluate the resulting throughput and IOPS, as shown in Figure 6. In this scenario, we configure `BUFFER_SIZE` to 8192, achieving an IOPS of 8000 with a bandwidth of 600 Mbps. With this IOPS, we can receive a packet every 125 microseconds, which is sufficient in our attack since it is at the same level as the microframe in USB 2.0 High-Speed mode [12] and bus interval period in USB 3.0 [40].

5.2 Camera Activity Detection

In this attack, we deploy the NIC Prober to monitor the camera traffic within the USB hub. Specifically, we connect both the Ethernet cable and the webcam, Camera1, to hub A2. The

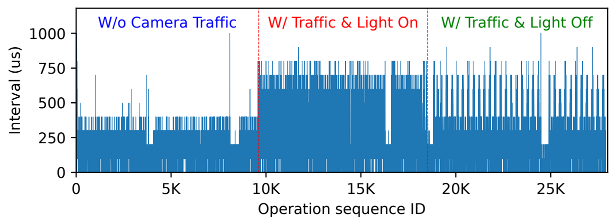


Figure 7: Camera Activity Detection by the NIC Prober

webcam is initially connected but not activated in a room with the light on. We then launch a camera application [74], initiating 1080p camera streaming at 30 frames per second (fps) over USB. Afterward, we turn off the room light. Figure 7 shows the time-series data captured by our NIC Prober, clearly indicating the transition points: when camera streaming begins and when the room light is turned off. The reason we observe different traffic patterns from the NIC prober is the USB webcam adopts H.264 video encoding standard. H.264 dynamically adjusts parameters based on scene complexity, motion, and other visual factors to compress the traffic while maintaining visual quality [37]. Moreover, the NIC Prober primarily captures camera traffic patterns due to the host's prioritization of isochronous data, leading to Scheduling Contention. Since the NIC Prober uses *bulk transfers*, it is affected by camera traffic that relies on *periodic transfers*, as bulk transfers can only be polled after periodic transfers are completed. This results in increased transfer delays caused by camera traffic, as illustrated in Figure 3.

Information inference from camera and video. Common video encoding standards such as MPEG-4 and H.264 produce distinct traffic patterns that correspond to scene changes and motion intensity. Based on this, *Walls Have Ears* [24] identifies streamed videos by analyzing network traffic. Prior work [38] demonstrates that information about basic daily activities can be inferred from the traffic patterns of encrypted video streams. Similarly, we show that an attacker can infer basic behaviors from camera traffic, as detailed in Subsection 7.3.

6 Attack C: Keystrokes Detection

In scenarios where deploying the probers on the host computer is unfeasible, the attacker can alternatively embed the prober within a USB device, which corresponds to Threat Model II. In this section, we demonstrate how to launch a HubBub attack from a USB peripheral. We assume that the peripheral and a keyboard are both connected to the host computer through the same USB hub, and the keyboard is being actively used by the victim.

6.1 Gadget Prober Design and Implementation

Due to USB being a host-centric communication protocol, the attacker within the USB device is unable to independently initiate a transaction and, consequently, cannot record the time interval during its execution. To address this limitation, we utilize a USB IN endpoint controlled by the device. Specifically, the attacker program continuously writes USB packets to this IN endpoint and records the time interval of the write operation. Successful writing to the IN endpoint occurs only when the USB host polls the device and the endpoint has available slots. If the host polls the device as expected, our prober will experience a time interval equal to the polling interval, either a frame (1 millisecond) or a microframe (125 microseconds), depending on the specific configuration. A time interval exceeding a frame or microframe indicates that the host failed to schedule the transfer within the expected period. Therefore, if the third type of bus contention, Shared-Structure Contention, occurs, our prober will experience higher time intervals. We present the detailed design of Gadget Prober in Algorithm 4 in the appendix.

Implementation. We build the Gadget Prober with a Raspberry Pi 4B, which features a USB Device Controller that supports On-The-Go (OTG) mode, i.e., it can behave as a USB device. We install Ubuntu 22.04 with kernel version `5.15.0-1077-raspi` on the Raspberry Pi and configure it as a USB gadget by setting up *Function Filesystem (FunctionFS)* interface. The configure script contains 49 lines of bash script. Subsequently, we develop a C program with 285 lines of code to implement the aforementioned mechanism. The details of this implementation are also presented in Algorithm 4. In detail, we configure the device for USB 2.0 (High-Speed) (`bcdUSB = 0x0200`), with a maximum power draw of `500 mA`, and set up FunctionFS (`ffs.isoc`) for isochronous transfers. We adjust the USB host to poll the endpoint every microframe, by setting `.bInterval` to 1.

We emulate the gadget prober as a USB 2.0 peripheral because the OTG mode of our Raspberry Pi is limited to USB 2.0. Nevertheless, USB 2.0 devices are compatible with USB 3.x hubs because a USB 3.0 hub includes a USB 2.0 hub, as illustrated in Figure 1.

Driver for Gadget Prober. We utilize `FunctionFS` to implement the Gadget Prober, enabling a more flexible approach to programming USB gadget devices compared to standard drivers like *g_mass_storage* or audio. Due to the lack of default drivers for `FunctionFS` gadgets on the USB host, we developed a simple USB host driver using *libusb*, which contains 131 lines of C code that only receives USB packets from the gadget device. This driver only polls the gadget without performing any malicious or suspicious operations. In real-world attack scenarios, adversaries can bypass the need for this custom driver by impersonating legitimate devices, such as BadUSB [50]. Since USB hosts support standard device classes like flash drives and microphones by default, such impersonation would trigger the loading of existing USB host device drivers. We leave the design of the Gadget prober that impersonates a known device to future work.

Timing function and parameters selection. For the timing function, we used POSIX API `clock_gettime()` with the `CLOCK_MONOTONIC` flag. In the default configuration of the Gadget Prober, we set the `TRANSFER_TYPE` to isochronous transfer and the `BUFFER_SIZE` to 100, which are defined in Algorithm 4. By using an isochronous endpoint, we can emulate the device as video and audio peripherals, such as a microphone, speaker, or camera. However, the attacker can also modify the `TRANSFER_TYPE` to bulk or interrupt as needed. For instance, if the victim device primarily uses bulk transfer, such as an SSD, we can switch the `TRANSFER_TYPE` to bulk to probe the traffic more accurately by exploiting Scheduling Contention. Because the host prioritizes periodic (interrupt and isochronous) traffic over asynchronous (bulk and control) traffic, isochronous transfers are less affected by the victim's traffic than bulk.

6.2 Keystrokes Detection

We attach the Gadget Prober and a standard keyboard, denoted as Keyboard1 in Table 8 in the appendix, to two downstream ports of the USB 3.0 internal hub on *Desktop B* (specifications in Table 7). Besides, Keyboard1 adopts the USB 1.1 protocol (Full-Speed and Low-Speed), which is common for HID devices, as they do not require high bandwidth to operate effectively.

We then typed word *HUBBUB* and extracted the intervals using the prober, specifically the `Intervals` from Algorithm 4. This sequence, illustrated in Figure 8, shows that most time intervals are around 125 µs. Because the prober can only write to the endpoint if the endpoint has available slots, i.e., after the host schedules the data transmission, the above results indicate that the host polls the endpoint for each microframe, which aligns with our configuration.

Figure 8 shows that we can distinguish keystrokes by identifying successive time intervals that exceed a microframe. The reason for the information leakage is as follows. Whenever a key press or release event is detected on the keyboard, it sends a USB packet with Interrupt transfer. Due to the keyboard operates with the USB 1.1 protocol, the higher-version USB controller will then send *Start-split* and *Complete-split* transactions to the USB hub, which are then translated and buffered by the Transaction Translator (TT) [12] within the USB 2.0 hub (note that our USB 3.0 hub also contains a USB 2.0 hub, as shown in Figure 1). The above translation introduces slight time delays, which are generally imperceptible to users but can be captured by the prober as shown in Figure 8. It is reasonable to infer that the time delay is attributable to the bus sharing of the TT structure and other hub functionalities, namely the third type of contention. Moreover, it is unlikely that the other two types of contention result in extra

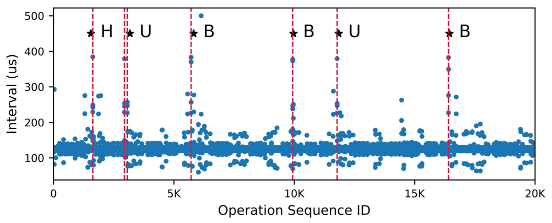


Figure 8: Keystrokes of *HUBBUB* Captured by the Gadget Prober

latency. The reasons are elaborated as follows. Firstly, the Gadget Prober utilizes isochronous transfer, which holds the highest priority within a microframe, as illustrated in Figure 3. Secondly, the bandwidth consumed by our prober is approximately 6.4 Mbps, with 100 bytes of data being transmitted per microframe. When combined with the keyboard's maximum throughput of 12 Mbps, the overall bandwidth usage is negligible in comparison to the 480 Mbps bandwidth of a USB 2.0 hub.

To filter keystrokes, we use a simple heuristic sliding window filter: detecting three consecutive high time intervals (over 170 microseconds) as a keystroke, which we mark with a vertical line in Figure 8. For comparison with ground truth, we mark the exact key typing time, calculated as the average of key press and release times, with a star and the letter typed. For letters *H*, *B*, and the second *U*, the release-to-press intervals are short, allowing us to detect only a single event. Besides, for the first *U*, the key press and release events are detected separately due to a longer release-to-press interval. Thus, we can accurately capture the timing of the keystrokes using the Gadget Prober. Additionally, we have confirmed that keystroke timings can also be detected on external hubs, as demonstrated in Subsection 7.4.

Text or password recovery. Earlier research [63, 68, 79] shows that passwords or text can be retrieved from keystroke timings using a Hidden Markov Model (HMM), which leverages statistical features. Specifically, different key pairs exhibit distinct inter-keystroke latency due to human typing habits; for instance, the latency between typing `v` and `o` is typically shorter than that between `v` and `b` [63]. By analyzing these timing patterns, an HMM can infer the most likely sequences of characters typed. Additionally, previous studies [59] demonstrate that they could train an RNN model to detect a specific sentence or a specific user. Similar to previous work, we also demonstrate that the recovered keystroke timings can further reveal the words typed, detailed in Subsection 7.4.

7 Evaluation

In this section, we introduce our experimental platform in Subsection 7.1. We then evaluate Attack A, B, and C in Subsection 7.2, 7.3, and 7.4 respectively. We then explore universality, transferability, and noise resilience of HubBub in Subsections 7.5 and 7.6. Finally, Subsection 7.7 discusses other potential attack vectors.

7.1 Experiment Platform

We use four desktop hosts, i.e., Desktop A, Desktop B, Desktop C, and Desktop D. Desktop A and Desktop B provide USB 3.0/3.1 ports for evaluating external USB 3.x hubs, while Desktop C and Desktop D are dedicated to cross-host transferability experiments. Their hardware details appear in Table 7 in the appendix. All attacks run on Chrome `130.0.6723.116`.

USB hubs. The external USB hubs evaluated were selected from the best-selling USB hubs on the market. Specifically, we searched for the keyword *USB hubs* on *Amazon* [1] and excluded unrelated items, such as charging stations. From the search results, we purchased 16 hubs for our study: 12 USB 3.0 hubs (A1-A12) and 4 USB 3.1 hubs (B1-B4). The cost of these hubs ranged from $7 to $40. Additionally, we evaluated monitor C1, a computer display that provides complete USB hub capabilities. Furthermore, each hub integrates an Ethernet NIC with 1000 Mbps bandwidth. We named each NIC according to the USB hub it is attached to; for instance, the NIC attached to A1 is referred to as ETH-A1, and so on. Our analysis also includes USB 2.0 hubs and internal USB hubs, designated as D1 and F1, respectively. Detailed specifications of all the USB hubs, NICs, and other USB devices are provided in Table 8 in the Appendix.

7.2 Website Inference with the SSD Prober

In this section, we use three experiments, i.e., E1, E2, and E3, to assess the performance of HubBub attack on Webpage Inference. Specifically, the SDD probers of E1 and E2 are deployed natively on the desktop to infer webpages under two settings. In E3, the prober is deployed within the VMM, which can still infer the visited webpage from the host.

For all subsequent experiments in this paper, unless otherwise specified, 80% of the dataset is randomly selected for training, with the remaining 20% reserved for testing. We pre-process the `Intervals` collected by our SSD prober before training the classifiers. *Accuracy* serves as our primary evaluation metric for the multi-class classification task, measuring the proportion of correct predictions. Additionally, we evaluate Top-3 accuracy, which considers a prediction correct if the true class is among the top three predicted labels.

The key findings from our evaluation are as follows: 1) the HubBub attack achieves high accuracy in website inference, particularly in the close-world setting, and 2) the SSD probers are capable of effectively monitoring USB hub traffic even within VMM isolation.

Table 1: Website Classification on HubBub-100 Dataset

Model	Top-1 Acc	Top-3 Acc	F1	Precision	Recall
HubAttGRU	**98.84%**	**99.53%**	**0.99**	**0.99**	**0.99**
DF [61]	58.57%	64.26%	0.64	0.73	0.59
DNNF [26]	41.68%	53.49%	0.42	0.42	0.42
TF [62]	5.81%	10.8%	0.06	0.09	0.06
Random Forest	24.12%	36.73%	0.23	0.23	0.24
XGBoost	25.88%	40.73%	0.26	0.27	0.26
MLP	6.82%	14.81%	0.07	0.07	0.07
Decision Tree	8.06%	9.86%	0.08	0.08	0.08
SVM	18.20%	28.99%	0.19	0.21	0.18

E1: Close-world setting. In a Close-world website fingerprinting setting, the classifier is trained and tested on the same set of websites. In this experiment, we utilize HubAttGRU to classify the HubBub-100 dataset, which was collected using A2, a USB 3.0 hub, to evaluate its ability to identify leading websites. As detailed in Table 1, HubAttGRU outperforms CNN-based and traditional machine learning classifiers. Given that the simpler DF model outperforms DNNF and TF, we only evaluate DF for CNN-based models in subsequent experiments. Moreover, among the ML-based classifiers, Random Forest and XGBoost yield the best performance. As shown in Table 1, HubAttGRU achieves Top-1 and Top-3 accuracies of 98.84% and 99.53% for 100 websites. Thus, we can effectively deduce the specific webpage accessed by the victim.

E2: Open-world setting. In an Open-world setting, the classifier needs to identify websites from both known and unknown sets. To evaluate, we follow the Open-world *Standard Mode* [55,62] in this experiment. Specifically, we combine the HubBub-100 and HubBub-OpenWorld datasets. Therefore, we have 101 categories, including 100 known websites and one open-world category representing all the unknown webpages. Our HubAttGRU classifier achieves Top-1 and Top-3 accuracies of 52.27% and 61.32%, respectively. Besides, for the open-world category, we achieve a precision of 87% and a recall of 89%. These results demonstrate that our model can effectively distinguish between websites within and outside the training data. By addressing class imbalance issues (we have around 5000 traces for the open-world category and 537 traces for normal categories), the overall accuracy in this setting could be further improved. One solution is to collect more traces for each website in the dataset, which we defer to future work.

E3: Virtual machine. We conduct this experiment to demonstrate that the SSD Prober functions effectively within VMM isolation. In this setting, a USB hub with an Ethernet interface is connected to *Desktop B*, with an SSD attached to the same hub and passed through to a virtual machine. We use *Kernel-based Virtual Machine (KVM)* [33] to host the virtual machine (VM) and *Libvirt* [39] to manage the VM. Their respective versions are `6.2.0` and `8.0.0`. The virtual machine is configured with 4 vCPUs and 4.0 GiB of memory, sharing the same operating system and kernel version as Desktop B.

We perform the Top-10 website classification task on A2 and B2, which are a USB 3.0 hub and a USB 3.1 hub, respectively. During the probing step, the SSD Prober collects 200 interval traces per website, totaling 2,000 traces per USB hub. We then train DF to infer the websites, achieving accuracies of 72.00% and 90.75% on A2 and B2, respectively. These results confirm that the SSD prober performs effectively within a virtual machine, thus compromising the VMM isolation assumptions.

7.3 Camera Activity Detection with the NIC Prober

To demonstrate the practicality of Attack B, we design an experiment to infer six basic daily activities captured by a camera. The setup is the same as in Subsection 5.2, with the NIC Prober deployed and camera traffic routed through USB Hub A2. Specifically, we collect 50 traces using the NIC Prober for the following behaviors: opening the camera using a camera application [74], and closing it through the same application; with the camera on, turning the light on, turning the light off, watching the screen while switching between two apps, and watching the screen while playing a five-second video.

After collecting the dataset, we preprocessed the traces using the method described in Subsection 4.2.2. As in the website fingerprinting task, 80% of the traces are used for training and 20% for testing. Given the dataset size, we use a classical machine learning approach, training a Random Forest classifier with 300 decision trees, each with a maximum depth of 10. The model is trained with a fixed random seed (`random_state=42`) to ensure reproducibility. The classifier achieves 85% accuracy on the test set, with a precision of 0.88, a recall of 0.85, and an F1-score of 0.83, demonstrating the feasibility of Attack B.

While the granularity of our prober is lower than that of prior work [24,38], which extracted fine-grained features such as encrypted packet sizes, the traffic fluctuations we capture remain sensitive to changes in brightness and reveal certain user activities, such as those targeted in this study.

7.4 Keystrokes Capture with the Gadget Prober

Keystrokes detection. We evaluated our attack in the setting described in Section 6.2, where the Gadget Prober records timing intervals from a USB keyboard connected through either an internal hub and A1, with 100 and 115 keystrokes typed, respectively. In total, 215 keys were typed across both setups. To obtain ground truth, we logged key press timestamps from `/dev/input/eventX` on Desktop B. We then applied the same filtering method as in Section 6.2 and grouped adjacent keystrokes occurring within 100 milliseconds as a single key press. We detected 198 keystrokes with 14 false positives, achieving a precision of 93.40% and a recall of 92.09%. These results confirm the effectiveness of the Gadget Prober in detecting keystrokes and also demonstrate its universality across different USB hubs.

Word inference. To demonstrate the practicality of this attack, we further evaluate the inference of 40 common English words typed on a standard USB keyboard. This setup is consistent with prior studies that evaluated 43 words [68] and 39 words [79], respectively. The dataset includes ten 3-letter words, ten 4-letter words, ten 5-letter words, and ten 6-letter words, with 50 traces of each word listed in Appendix B. Then, we converted the captured timestamps into inter-keystroke delays and padded them to a uniform length.

Using 80% of the data for training the Random Forest classifier used in Subsection 7.3, we achieved 72.5% and 90.75% Top-1/Top-3 accuracy on the test set. Specifically, the Top-1 accuracies for 3-, 4-, 5-, and 6-letter words were 46%, 70%, 89%, and 85%, respectively. As expected, longer words contain more inter-keystroke delays, providing richer features for classification. Note that these distributions also reflect an individual's typing patterns. Therefore, the accuracy might decrease when the model is used to infer words typed by a different person. In summary, our evaluation demonstrates that Attack C has the potential to reveal sensitive information such as typed words.

7.5 Attack Generality and Transferability

In this subsection, we revisit Attack A to evaluate the robustness and impact of HubBub on various USB hubs. Our evaluation demonstrates that: 1) HubBub affects all 15 external USB 3.x hubs in our experiments; 2) internal and USB 2.0 hubs are also vulnerable to HubBub; 3) HubBub attacks are transferable between different USB hubs; and 4) HubBub attacks are transferable between different hosts computers.

External USB 3.x hubs. In this experiment, we use the HubBub-45 dataset to evaluate the website inference accuracy. Table 2 shows the accuracy of HubAttGRU, DF, and Random Forest on different USB hubs. Compared to the evaluation on the HubBub-100 dataset, we slightly reduced the model parameters of HubAttGRU to adapt to the smaller dataset size. Besides, we trained both HubAttGRU and DF for 200 epochs. While more epochs may further enhance accuracy, we leave this for future work because the current results sufficiently illustrate HubBub's impact on various USB 3.x hubs.

Our findings show that for most USB 3.x hubs, HubAttGRU can accurately infer websites. For hubs A1, A2, A4, A5, A7, A10, A11, and B2, we can achieve over 80% accuracy. Even for the hub with the lowest accuracy, B1 at 11.61%, the performance is still significantly better than the accuracy of a random classifier (with an accuracy of $\frac{1}{45}$ or 2.22%), indicating that information of the victim can still be inferred, albeit at a coarser level. Therefore, we assert that most USB hubs are vulnerable to the HubBub attack, and all the hubs in our testing are affected by HubBub. Moreover, the reason why the accuracy of HubBub varies on different USB hubs might be due to the priority of USB ports within the USB hub. Because both the NIC and SSD traffic use bulk transfer, their priority of scheduling in the hub is theoretically the same. However, due to specific USB hub implementations, the USB hub may assign a lower priority to the NIC port than a normal port used by the USB SSD. For example, the NIC port of B1 may have lower priority than that of other hubs. This situation causes the traffic of our SSD prober to be less influenced, which in turn causes it to capture fewer patterns.

Table 2: Website Inference Accuracy on HubBub-45 Dataset

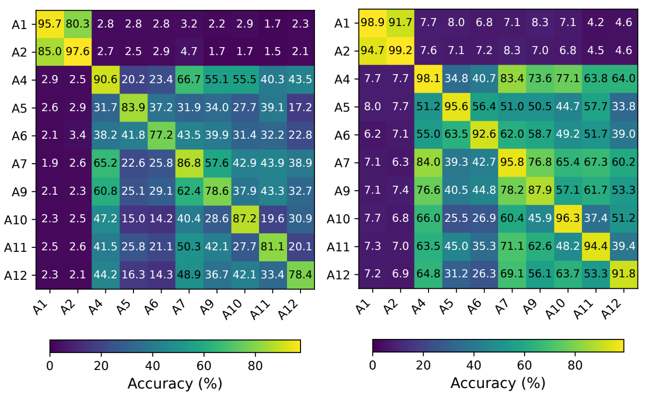
Hub	Brand	Spec	HubAttGRU	DF	Random Forest
A1	UNI	USB 3.0	95.67%	93.86%	82.1%
A2	UGREEN	USB 3.0	97.56%	95.56%	86.61%
A4	TPLINK	USB 3.0	90.56%	85.06%	40.3%
A5	ABLEWE	USB 3.0	83.89%	83.89%	45.6%
A6	ACEELE	USB 3.0	77.17%	77.28%	39.3%
A7	OYLIAN	USB 3.0	86.83%	80.28%	38.4%
A9	Fophmo	USB 3.0	78.56%	74.22%	35.1%
A10	ACER	USB 3.0	87.17%	82.56%	38.3%
A11	WAVLINK	USB 3.0	81.11%	84.17%	45.9%
A12	UtechSmart	USB 3.0	78.44%	83.83%	39.0%
B1	RSHTECH	USB 3.1	11.61%	9.22%	7.7%
B2	INATECK	USB 3.1	94.06%	92.97%	80.7%
B3	WAVLINK	USB 3.1	72.17%	69.72%	56.3%
B4	Getatek	USB 3.1	29.06%	20.83%	11.9%
C1	Dell	USB 3.0	57.83%	51.39%	20.3%

Internal hub and USB 2.0 hub. Firstly, we expand the evaluation to a USB 3.0 internal hub, which is the root hub of *Desktop A* (F1 in Table 8 in the Appendix). Specifically, we connect SSD3 and ETH-D1, the USB NIC, to F1. After training for 200 epochs, we achieve a classification accuracy of 79.33% with HubAttGRU for the top 45 websites.

In the USB 2.0 hub evaluation, we use D1 as the hub and ETH-D1 as the USB NIC. We configure SSD1 with an `IO_SIZE` setting of 4096. Then, we evaluate the top 10 websites and collect 200 traces of approximately 5 seconds for each. We choose DF as the classifier, which fits this data scale better than HubAttGRU. In our experiment, we achieved an accuracy of 91.25%, which shows that HubBub are also effective on internal hubs and USB 2.0 hubs.

USB hub transferability. In this experiment, we evaluate the cross-hub accuracy of HubAttGRU across all USB 3.0 hubs listed in Table 2. Assuming we train on hub X and test on hub Y, we use 80% of the traces from HubBub-45-A[X] to train the model for 200 epochs and evaluate it on 20% of traces from HubBub-45-A[Y] as the test set. As shown in Figure 9, we list all the cross-hub transferability Top-1/Top-3 accuracy in two heatmaps. We then adopt three metrics to evaluate the transferability: Top-1/Top-3 accuracy, the multiplicative gain over random guessing accuracy (2.22% for a 45-class classifier), and the accuracy drop compared to the classifier trained directly on traces from hub Y.

After analysis, we observe two distinct groups of hubs for



(a) Top-1 Accuracy (b) Top-3 Accuracy

Figure 9: Cross-hub Transferability of HubAttGRU Top-45 Website Inference. The X-axis Represents the Training Hub

transferability. *Group 1* includes A1 and A2, exhibiting a strong mutual transferability, with Top-1/Top-3 accuracy exceeding 80% and 90%. The average accuracy improvement over random guessing is 36.18×, and there is only a 14.5% accuracy drop. *Group 2* includes hub A4 to A12, showing cross-hub Top-1/Top-3 accuracy ranging from 14.2%/25.5% to 66.7%/84%, with an average of 36.11%/54.51%. The average improvement over random guessing is around 16.25×, and the average accuracy drop is 56.4%. In the worst case, nevertheless, the accuracy remains 4.2 times higher than random guessing.

In comparison, the transferability across groups is low. The Top-1/Top-3 accuracy is approximately 2.50%/6.94%, indicating little to no improvement over random guessing. We further investigated the potential reason by visualizing traces from both groups. We observed that traces collected from *Group 1* are notably stable, although with some noise, whereas traces from *Group 2* are more variable and share a similar noise pattern. For example, in *Group 2*, some hubs exhibit significantly longer delay sequences during the first second, while others show extended delays during the middle portion of the trace. This also explains why the models achieve higher accuracy on A1 and A2 in Table 2. This characteristic may stem from power-saving and traffic management policies implemented in the firmware or microcontroller of each hub. We leave the exploration of improving transferability across groups to future work, which may benefit from more precise temporal alignment and enhanced noise filtering during pre-processing.

USB host transferability. In this evaluation, we demonstrate HubBub's host transferability by training and testing HubAttGRU on different computers. We collected 200 traces for the 45 websites each from Desktop A, Desktop C, and Desktop D using hub A1. Each dataset was split into 80% for training and 20% for testing the model. We observed that varying CPU frequencies across hosts affect `rdtscp`-based timing,

Table 3: Cross-host Transferability of HubAttGRU Top-45 Website Inference (Top-1/Top-3, %)

Test / Train	Desktop A	Desktop C	Desktop D
Desktop A	83.39 / 94.33	30.56 / 54.83	29.22 / 53.83
Desktop C	19.44 / 33.17	84.50 / 96.00	18.00 / 32.94
Desktop D	35.44 / 60.50	31.56 / 53.00	89.67 / 98.44

so we applied simple normalization during preprocessing. The Top-1/Top-3 accuracy of cross-host website inference is listed in Table 3.

The results show an average Top-1/Top-3 accuracy of 27.37% and 48.04%, respectively, representing an average 12.31× improvement over random guessing and a 68.21% drop compared to same-host performance. The results demonstrate that models trained on one computer can remain effective on others despite some accuracy degradation.

We further investigated the cause of the accuracy drop and found that variations in noise patterns across hosts contributed to the decline. We leave cross-host inference improvements to future work, potentially through more effective noise filtering or adversarial noise injection to enhance robustness.

7.6 Effective Bandwidth and Noise Resilience

We first design three covert-channel attacks targeting the three probers: *CC1 (Covert Channel 1)* for the SSD prober, *CC2* for the NIC prober, and *CC3* for the Gadget prober. We adjust the number of packets sent by the sender to induce a noticeable delay in the traces captured by the prober, which we use to transmit bit `1`. To transmit bit `0`, the sender pauses for the same duration. We then measure their ideal bandwidths, which are 500 bit/s, 4 bit/s, and 5 bit/s, respectively. Detailed designs are provided in Appendix C.

Effective Bandwidth. For each attack, we analyze a sequence of 120 transmitted bits from the receiver's trace and observe 0, 1, and 3 bit errors for *CC1*, *CC2*, and *CC3*, respectively. This corresponds to bit error rates (BER) of 0%, 0.83%, and 2.5%. We treat all channels as symmetric, as both `1` and `0` may be misclassified as each other. The effective bandwidth (or channel capacity), denoted by $C = R\big(1 - H_2(p)\big)$ [14], is calculated using the ideal bit rate R and the binary entropy function $H_2(p)$, where p is the BER. Based on this, the effective bandwidths of *CC1*, *CC2*, and *CC3* are 500 bit/s, 3.72 bit/s, and 4.16 bit/s, respectively.

Bandwidth Degradation Under Noise. To evaluate noise resilience, we connect SSD2 to the USB hub and inject noise by performing file access operations from the host. Specifically, our noise injection program reads 256 KB from one of ten randomly selected 200 MB files. All files are opened with the `O_DIRECT` flag to bypass the page cache.

We simulate four noise levels by varying the file access frequency: 1000 ms (Level 1), 100 ms (Level 2), 10 ms (Level 3), and 1 ms (Level 4), with the baseline (no injected noise) denoted as Level 0. For each level, we analyze a 120-bit transmission sequence from all three covert-channel attacks and compute the effective bandwidth, as shown in Table 4. For *CC1*, we observe an apparent decline in capacity as noise increases. At Level 4 (1 ms access interval), the bandwidth drops to 17.07 bit/s, preserving only 3.4% of its original capacity. For *CC2* and *CC3*, the signal becomes indistinguishable from noise at Level 3 and Level 2, respectively, and we therefore mark their effective bandwidths as N/A.

Table 4: Effective Bandwidth (Eff) under Noise

Noise Level	CC1 Eff (bit/s)	CC2 Eff (bit/s)	CC3 Eff (bit/s)
Level 0	500.00	3.72	4.16
Level 1	438.85	3.16	1.23
Level 2	438.85	3.00	N/A
Level 3	293.09	N/A	N/A
Level 4	17.07	N/A	N/A

7.7 Other Attacking Combinations

In our three demonstrated attacks, each primitive was initially evaluated against a single target. To illustrate broader applicability, we assess the feasibility of all three targets across all probers, as summarized in Table 5, where filled circles indicate confirmed attacks, empty circles denote infeasible ones, and half-filled circles represent potential attacks with preliminary feasibility testing.

Table 5: Capability Matrix of Probers against Targets

Prober\Target	Website Fingerprinting	Camera	Keystrokes
SSD Prober	●	◐	○
NIC Prober	●	●	○
Gadget Prober	◐	◐	●

Attack A, B, and C correspond to the diagonal entries in Table 5 and are marked with filled circles. Additionally, we find that the NIC prober can leak information about visited websites. To validate this, we connect an additional USB NIC to Hub A1 and collect 200 traces across 10 websites using our prober. Using an 80:20 train-test split, we train a Random Forest classifier and achieve an accuracy of 73.3%.

We further show that the SSD Prober can potentially leak camera-related activity. Specifically, with `IO_SIZE` set to 8192, it can distinguish traces with the camera on versus off due to congestion caused by the camera traffic. We also found that the Gadget prober may leak information about website visits and camera activity. For website visits, the prober is sensitive to Ethernet traffic from YouTube video preloading, which increases USB endpoint write delays of the Gadget prober. Similarly, enabling the camera introduces additional USB traffic on the hub, resulting in measurable delays of the Gadget prober.

Unfortunately, we could not recover keystrokes from the USB keyboard using the SSD or NIC prober. We suspect the interrupt traffic generated by keystrokes is too small relative to the probers' bulk traffic and may even fall below the noise level in their traces.

8 Potential Mitigations

HubBub attacks might be mitigated in the following ways. First, we can restrict the use of high-precision timing functions. . To mitigate this, browsers could enhance isolation mechanisms by requiring websites to request permission before accessing this or similar functions. Besides, our SSD Prober uses `rdtscp` or `rdtsc` to measure CPU cycles, which could also be restricted for untrusted users [25]. Additionally, reducing the resolution of timing functions may help lower the attack's accuracy. For example, starting from Firefox 59, the default resolution of `performance.now()` was reduced to 2 milliseconds.

Second, USB hub manufacturers can upgrade firmware to implement countermeasures against HubBub. One approach is to introduce random delays in the delivery of USB packets. However, this method would reduce bandwidth and increase latency, trading performance for enhanced security. Besides, a hardware-level detection approach can be implemented by integrating a small, programmable security controller into the USB hub. This on-hub controller passively monitors traffic on each downstream port, parsing packets and applying rule-based or machine learning logic to detect malicious flows. Upon identifying suspicious activity, it can initiate a standard control transfer to alert the host.

Third, USB bandwidth isolation can be enforced on the host by reserving a dedicated portion of one or more micro-frames for each device, ensuring that its packets are polled exclusively within those slots. It can be implemented by updating scheduling policies similar to those proposed for improving power efficiency in USB hubs [57].

9 Related Works

In both our threat models, alternative attacks exist. Below, we discuss their respective advantages and limitations and compare them to HubBub. In addition, we list other USB attacks in Appendix D.

Threat Model I. First, *Invisible Probe* [68] exploits PCIe switch and PCH congestion to infer secrets such as website fingerprints, passwords, and machine learning models. Like HubBub, it requires access to a peripheral. However, due to PCIe's higher bandwidth, inducing sufficient congestion typically demands high-speed devices. Moreover, their attacks are tailored for cloud environments, e.g., requiring a remote machine connected to the victim host via *RDMA NIC* to launch attacks. In contrast, HubBub targets typical desktops and laptops and does not require high bandwidth peripherals like RDMA. Second, *SMASH* [16] and *Rowhammer.js* [23] break browser isolation by exploiting Rowhammer attacks. However, they rely on vulnerable DRAM modules. In contrast, HubBub does not require vulnerable DRAM. Third, Cache side channels [35, 42–44, 51, 76, 78] are pervasive and can leak sensitive information such as private keys. In contrast, HubBub does not rely on prior architectural knowledge, nor does the attacker need to manage low-level details such as identifying which cache set buffers a given memory access. Finally, *MeshUp* [71], *Lord of Ring* [52], and *Don't Mesh Around* [15] are contention-based side-channel attacks that leak information such as RSA private keys, from CPU interconnects like Rings, Mesh, and UPI. Unlike HubBub, their attacks rely on knowledge of CPU core topology and cache structure, similar to the above-mentioned cache-based attacks.

Threat Model II. First, *USBhubleakage* [66] constructed a harmful peripheral device capable of intercepting confidential USB traffic from neighboring paths, which leaks keystrokes from the USB hub. However, their attack can only succeed on USB 1.x/2.0 hubs. Second, prior to USB 3.0, a device plugged into a hub could eavesdrop on traffic intended for any other downstream port [48], due to all downstream packets being broadcast to every port. USB 3.0 removes the threat by replacing broadcast forwarding with unicast routing, so the attack no longer works on modern hubs. In contrast to the attacks described above, HubBub also targets USB 3.x hubs. Third, Key injection attacks like *USB Rubber Ducky* [6] and *USBImpostor* [18] can masquerade as keyboards and issue arbitrary commands on the host, but they are relatively easy to detect. For example, macOS prompts for user authentication when a new keyboard is connected, which hinders spoofed devices. By contrast, HubBub evades USB-peripheral defenses and remains invisible to the user.

10 Conclusion

In summary, our work reveals that when multiple USB peripherals are connected to a single USB hub, bus contention can arise, which poses a privacy risk. By leveraging USB bus contention, we design HubBub which can extract sensitive information from USB hubs. Our evaluations demonstrate that attackers can infer visited webpages, activities captured by the camera, and keystrokes via HubBub attacks. We hope that this work raises awareness and inspires the implementation of protective measures against HubBub attacks.

Acknowledgment

This work was supported in part by the National Science Foundation (Grant No. CNS-2145744). We thank the anonymous reviewers for their constructive feedback and our colleagues for their helpful suggestions.

11 Ethics considerations

We have reported our findings to relevant stakeholders and the USB Implementers Forum (USB-IF). Besides, we conducted all experiments within our controlled environment, and no attacks were carried out against any third-party environment.

12 Open Science Policy

To facilitate reproducibility and further research, we released the source code for three probers and the main classifier, along with the HubBub Website Fingerprinting Dataset. The artifact can be accessed at https://doi.org/10.5281/zenodo.15581471.

References

[1] `https://www.amazon.com/s?k=USB+hub&s=exact-aware-popularity-rank`. Accessed: April 14, 2024.

[2] Cynthion — all-in-one usb test instrument, 2023. Accessed 18 May 2025.

[3] Alexa top websites - expireddomains.net, 2024.

[4] Trending websites - semrush, 2024.

[5] Olga Angelopoulou, Seyedali Pourmoafi, Andrew Jones, and Gaurav Sharma. Killing your device via your usb port. In *Proceedings of the Thirteenth International Symposium on Human Aspects of Information Security & Assurance (HAISA 2019)*, pages 61–72. The Centre for Security, Communications and Network Research (CSCAN), 2019.

[6] Lakshay Arora, Narina Thakur, and Sumit Kumar Yadav. Usb rubber ducky detection by using heuristic rules. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 156–160. IEEE, 2021.

[7] Jens Axboe. Efficient io with io_uring. *URL https://kernel. dk/iouring. pdf*, 2019.

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

[9] Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, NEC Corporation, ST-NXP Wireless, and Texas Instruments. Universal serial bus specification 3.0 revision 1.0. Specification, USB Implementers Forum, November 2008.

[10] Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, Renesas Corporation, ST-Ericsson, and Texas Instruments. Universal serial bus specification 3.1 revision 1.0. Specification, USB Implementers Forum, July 2013.

[11] Compaq, Digital Equipment Corporation, IBM PC Company, Intel, Microsoft, NEC, and Northern Telecom. Universal serial bus specification revision 1.0. Specification, USB Implementers Forum, January 1996.

[12] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. Universal serial bus specification revision 2.0. Specification, USB Implementers Forum, April 2000.

[13] Compaq, Intel, Microsoft, and NEC. Universal serial bus specification revision 1.1. Specification, USB Implementers Forum, September 1998.

[14] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[15] Miles Dai, Riccardo Paccagnella, Miguel Gomez-Garcia, John McCalpin, and Mengjia Yan. Don't mesh around:{Side-Channel} attacks and mitigations on mesh interconnects. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2857–2874, 2022.

[16] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. {SMASH}: Synchronized many-sided rowhammer attacks from {JavaScript}. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1001–1018, 2021.

[17] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[18] Robert Dumitru, Daniel Genkin, Andrew Wabnitz, and Yuval Yarom. The impostor among US(B): Off-Path injection attacks on USB communications. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5863–5880, Anaheim, CA, August 2023. USENIX Association.

[19] Zakir Durumeric. Crux top lists, 2023.

[20] Monta Elkins. Universal rf usb keyboard emulation device urfuked, June 2010.

[21] Ian Fette and Alexey Melnikov. The websocket protocol. Technical report, 2011.

[22] Daniel Gruss, David Bidner, and Stefan Mangard. Practical memory deduplication attacks in sandboxed javascript. In *Computer Security–ESORICS 2015: 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I 20*, pages 108–122. Springer, 2015.

[23] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer. js: A remote software-induced fault attack

in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 300–321. Springer, 2016.

[24] Jiaxi Gu, Jiliang Wang, Zhiwen Yu, and Kele Shen. Traffic-based side-channel attack in video streaming. *IEEE/ACM Transactions on Networking*, 27(3):972–985, 2019.

[25] Part Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, 2(11):1–64, 2011.

[26] Maohua Guo, Jinlong Fei, and Yitong Meng. Deep nearest neighbor website fingerprinting attack technology. *Security and Communication Networks*, 2021(1):5399816, 2021.

[27] Mordechai Guri, Matan Monitz, and Yuval Elovici. Usbee: Air-gap covert-channel via electromagnetic emission from usb. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 264–268, 2016.

[28] Apple Inc., Hewlett-Packard Inc., Intel Corporation, Microsoft Corporation, Renesas Corporation, STMicroelectronics, and Texas Instruments. Universal serial bus specification 3.1 revision 1.0. Specification, USB Implementers Forum, September 2017.

[29] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. A shared cache attack that works across cores and defies vm sandboxing–and its application to aes. In *2015 IEEE Symposium on Security and Privacy*, pages 591–604. IEEE, 2015.

[30] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Wait a minute! a fast, cross-vm attack on aes. In *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings 17*, pages 299–319. Springer, 2014.

[31] Zhaoxin Jin, Tianbo Lu, Shuang Luo, and Jiaze Shang. Transformer-based model for multi-tab website fingerprinting attack. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1050–1064, 2023.

[32] Martin Johns. On javascript malware and related threats: Web page based attacks revisited. *Journal in Computer Virology*, 4(3):161–178, 2008.

[33] Kernel-based Virtual Machine Project. *KVM: Kernel-based Virtual Machine*, Year of Publication or Access. https://www.linux-kvm.org.

[34] David Kierznowski. Badusb 2.0: Exploring usb man-in-the-middle attacks.

[35] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7):93–101, 2020.

[36] David Kushner. The real story of stuxnet. *ieee Spectrum*, 50(3):48–53, 2013.

[37] Jeehong Lee, IlHong Shin, and HyunWook Park. Adaptive intra-frame assignment and bit-rate estimation for variable gop length in h. 264. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(10):1271–1279, 2006.

[38] Hong Li, Yunhua He, Limin Sun, Xiuzhen Cheng, and Jiguo Yu. Side-channel information leakage of encrypted video stream in video surveillance systems. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[39] Libvirt Developers. *libvirt: The virtualization API*, Year of Publication or Access. https://libvirt.org.

[40] Chong Han Lim, Bakhtiar Affendi bin Rosdi, and Chee Fai Yap. Synchronization of multiple usb 3.0 devices using isochronous timestamp packet. *Computer Standards & Interfaces*, 49:22–33, 2017.

[41] Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice, and Stefan Mangard. Practical keystroke timing attacks in sandboxed javascript. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*, pages 191–209. Springer, 2017.

[42] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. {ARMageddon}: Cache attacks on mobile devices. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 549–564, 2016.

[43] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[44] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE, 2015.

[45] Hao Liu, Riccardo Spolaor, Federico Turrin, Riccardo Bonafede, and Mauro Conti. Usb powered devices: A survey of side-channel threats and countermeasures. *High-Confidence Computing*, 1(1):100007, 2021.

[46] man.archlinux.org. io_uring_setup(2) - linux manual page. https://man.archlinux.org/man/io_uring_setup.2.en, 2019. Accessed: 2025-05-17.

[47] Market.us. Usb hub market size and growth report. https://market.us/report/usb-hub-market/, 2023. Accessed: November 7, 2024.

[48] Matthias Neugschwandtner, Anton Beitler, and Anil Kurmus. A transparent defense against usb eavesdropping attacks. In *Proceedings of the 9th European Workshop on System Security*, pages 1–6, 2016.

[49] Tao Ni, Yongliang Chen, Weitao Xu, Lei Xue, and Qingchuan Zhao. Xporter: A study of the multi-port charger security on privacy leakage and voice injection. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2023.

[50] Karsten Nohl and Jakob Lell. Badusb-on accessories that turn evil. *Black Hat USA*, 1(9):1–22, 2014.

[51] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology–CT-RSA 2006: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2005. Proceedings*, pages 1–20. Springer, 2006.

[52] Riccardo Paccagnella, Licheng Luo, and Christopher W Fletcher. Lord of the ring (s): Side channel attacks on the {CPU}{On-Chip} ring interconnect are practical. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 645–662, 2021.

[53] PCI-SIG. *PCI Express® Base Specification*. PCI-SIG, revision 4.0 version 0.3 edition, 2 2014. https://astralvx.com/storage/2020/11/PCI_Express_Base_4.0_Rev0.3_\February19-2014.pdf.

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[55] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[56] Gaurav Shah, Andres Molina, Matt Blaze, et al. Keyboards and covert channels. In *USENIX Security Symposium*, volume 15, page 64, 2006.

[57] Bikrant Das Sharma, Abdul Rahman Ismail, and Chris Meyers. Power savings in usb hubs through a proactive scheduling strategy. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–7. IEEE, 2023.

[58] Seungwon Shin and Guofei Gu. Conficker and beyond: a large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 151–160, 2010.

[59] Laurent Simon, Wenduan Xu, and Ross Anderson. Don't interrupt me while i type: Inferring text entered through gesture typing on android keyboards. Privacy Enhancing Technologies Symposium Advisory Board, 2016.

[60] Arjun Singh, Pushpa Choudhary, et al. Keylogger detection and prevention. In *Journal of Physics: Conference Series*, volume 2007, page 012005. IOP Publishing, 2021.

[61] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1928–1943, 2018.

[62] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1148, 2019.

[63] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on {SSH}. In *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[64] Branislav Sredojev, Dragan Samardzija, and Dragan Posarac. Webrtc technology overview and signaling solution design and implementation. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 1006–1009. IEEE, 2015.

[65] StoredBits. Why do ssds have slower data write speeds? https://storedbits.com/why-do-ssds-have-slower-data-write-speed/, 2024. Accessed: November 7, 2024.

[66] Yang Su, Daniel Genkin, Damith Ranasinghe, and Yuval Yarom. USB snooping made easy: Crosstalk leakage attacks on USB hubs. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1145–1161, Vancouver, BC, August 2017. USENIX Association.

[67] Peter Szor. Duqu–threat research and analysis. *McAfee Labs*, 2011.

[68] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. Invisible probe: Timing attacks with pcie congestion side-channel. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 322–338, 2021.

[69] Stella Vouteva, Ruud Verbij, and Jarno Roos. Feasibility and deployment of bad usb. *University of Amsterdam, System and Network Engineering Master Research Project*, 2015.

[70] Jim Walter. Flame attacks": Briefing and indicators of compromise. *McAfee Labs Report*, 1:43, 2012.

[71] Junpeng Wan, Yanxiang Bi, Zhe Zhou, and Zhou Li. Meshup: Stateless cache side-channel attack on cpu mesh. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1506–1524, 2022.

[72] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

[73] Yuanda Wang, Hanqing Guo, and Qiben Yan. Ghosttalk: Interactive attack on smartphone voice system through power line. *arXiv preprint arXiv:2202.02585*, 2022.

[74] Jonas Westman. camera – gnome virtual camera implementation. `https://gitlab.gnome.org/jwestman/camera`, 2020.

[75] Zhen Wu, Peng Hu, Shuangyue Liu, and Tao Pang. Attention mechanism and lstm network for fingerprint-based indoor location system. *Sensors*, 24(5):1398, 2024.

[76] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 888–904. IEEE, 2019.

[77] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S Balagani. On inferring browsing activity on smartphones via usb power analysis side-channel. *IEEE Transactions on Information Forensics and Security*, 12(5):1056–1066, 2016.

[78] Yuval Yarom and Katrina Falkner. {FLUSH+RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In *23rd USENIX security symposium (USENIX security 14)*, pages 719–732, 2014.

[79] Kehuan Zhang and XiaoFeng Wang. Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems. In *USENIX Security Symposium*, volume 20, page 23, 2009.

[80] Wu Zhenyu, Xu Zhang, and H Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *USENIX Security symposium*, pages 159–173, 2012.

A HubBub Website Fingerprinting Dataset

HubBub-100 Dataset. We built this dataset to evaluate HubBub in inferring globally leading websites. To minimize experimental bias, we selected two reputable sources: Alexa Top Websites [3] and Semrush [4]. After removing duplicates, we finalized a list of 100 websites. The environment setting is the same as Subsection 4.1. We connect SSD1 to Desktop A via a USB 3.0 hub, A2. For each of the 100 websites, we conducted 537 individual traces, collecting the resulting time-series data with the SSD Prober for around 5 seconds. As a result, HubBub-100 comprises 53,700 traces.

HubBub-45 Dataset. We created this subset to study the impact of HubBub on various USB hubs. HubBub-45 comprises data from 15 hubs spanning USB versions 2.0, 3.0, and 3.1, with all USB hubs listed in Table 2 included. We excluded A3 and A8 due to their instability and frequent connection losses during data collection. Consequently, the HubBub-45 dataset consists of 15 subsets named HubBub-45-A1, HubBub-45-A2, and so forth. In addition, SSD1 was used to test USB 3.0 hubs (A1-A12), SSD3[3] was employed for USB 3.1 hubs (B1-B4), and SSD2 was used to evaluate C1. Same as HubBub-100 Dataset, we collect data on Desktop A. Due to data collection being time-intensive, e.g., approximately 4 days for HubBub-100 on one USB hub, we scaled down the data for each HubBub-45 subset to about one-fifth of the size of HubBub-100. Specifically, we selected only 45 websites and collected 200 traces per website. Since the purpose of this dataset is to demonstrate the feasibility of HubBub across different USB hubs effectively, our objective can still be achieved. Consequently, HubBub-45 consists of a total of 135,000 traces (15 USB hubs × 45 websites × 200 traces).

HubBub-OpenWorld Dataset. To evaluate the Open-World scenario, we use 2,818 valid websites crawled by *Chrome Top Million Websites (CrUX)* [19]. We collect two traces for each website in the same setting with the HubBub-100 Dataset. Consequently, this dataset consists of 5,636 traces.

[3]The maximum throughput of SSD3 is 3,000 MB/s.

B Word List

and bus cup dog key man pig sky sun two city game give hand life name park play time this angle chair crown light quick south super about audio candy double island jacket people planet quench random spring rocket search

C Covert-channel Designs

In *CC1* (Covert Channel 1), we use the SSD prober as the receiver. The sender program is deployed within the Local Area Network (LAN), and transmits UDP packets to the target machine's NIC, which is connected via the USB hub. In the sender, a buffer size of 256 KB is used to represent a binary `1`, and a 2 ms sleep interval denotes a binary `0`, yielding a stable bit transmission rate of 500 bit/s.

In *CC2*, the gadget prober acts as the sender and the NIC prober as the receiver. To transmit bits reliably, the gadget prober sends continuous interrupt packets for 250 ms to represent a binary `1` and remains idle for 250 ms to represent a binary `0`, resulting in a bandwidth of approximately 4 bit/s.

In *CC3*, we use a Cynthion device [2] to inject keystrokes as the sender, and the gadget prober serves as the receiver. The sender issues three keystrokes to represent a binary `1` and waits 200 ms to represent a binary `0`, achieving a bandwidth of around 5 bit/s.

D Other USB Attacks

We classify USB-based attacks into invasive and non-invasive categories. **Invasive attacks** are designed to introduce keystrokes [6,18,20] or code fragments [36,58,67,70] into the USB host, or inflict damage on it [5]. **Non-invasive attacks** typically passively and covertly intercept confidential data from the USB channel, making them difficult to detect. For example, *USB Key Logger* [60] records keystrokes by attaching a device to the USB communication path. *BADUSB 2.0* [34] accomplishes identical keystroke recording by fabricating a custom USB cable and initiating a Man-in-the-Middle assault. *JitterBug* [56] are instruments that can discreetly exfiltrate confidential data from input devices (e.g., keyboard) by inserting minor, nearly undetectable pauses following each keystroke, thus embedding data into the timing of the subsequent network traffic. Relying on an unmodified USB device without RF transmitters, *USBee* [27] encodes and transmits data through electromagnetic emissions from the USB connector's data bus. The electromagnetic signal could then be decoded by a nearby receiver. Besides, power side-channel attacks [45, 49, 73, 77] can also be launched against USB hosts or devices. This involves a distinct method of examining power consumption, distinct from our work. HubBub can be classified as a Non-invasive USB attack as well.

E Supplementary Algorithms and Tables

Algorithm 2 NIC Prober

```
Input: buffer_size, access_num
Output: Intervals
1:  Intervals = [ ]
2:  Conn ← CONNECTIONREQUEST(server, WebSocket)
3:  Buffer = MEMORYALLOC(buffer_size)
4:  TS ← GETTIME()                                ▷ TS = timestamp
5:  for i ← 1 to access_num do
6:      Packet ← RECIVE(Conn, buffer_size)
7:      Intervals.APPEND(GETTIME() - TS)
8:      TS ← GETTIME()
9:  end for
10: MEMORYRELEASE(Buffer)
11: CLOSE(Conn)
```

Algorithm 3 NIC Prober (Server Side)

```
Input: buffer_size, access_num
Output: None
1: Conn ← CONNECTIONRESPOSE(client, WebSocket)
2: Buffer = MEMORYALLOC(buffer_size)
3: for i ← 1 to access_num do
4:     Buffer ← Payload
5:     SEND(Conn, Buffer)
6:     AWAIT(client, confirmation)
7: end for
8: MEMORYRELEASE(Buffer)
9: CLOSE(Conn)
```

Algorithm 4 Gadget Prober

```
Input: transfer_type, buffer_size, access_num
Output: Intervals
1:  CREATE EP&IN(transfer_type)
2:  Intervals ← [ ]
3:  Buffer ← MEMORYALLOC(buffer_size)
4:  TS ← GETTIME()                                ▷ TS = timestamp
5:  for $i$ ← 1 to access_num do
6:      WRITE(EP, buffer)                         ▷ Write buffer to EP
7:      Intervals.APPEND(GETTIME() − TS)
8:      TS ← GetTime()
9:  end for
10: MEMORYRELEASE(Buffer)
```

Table 6: Architecture and Dimensions of HubAttGRU Model

Parameters/Layer	Embedding	GRU	Attention	FC1	FC2	FC3
Input Dimension	1	384	384	384	256	128
Output Dimension	384	384	384	256	128	100

Table 7: Platform Specifications

	Desktop A	**Desktop B**	**Desktop C**	**Desktop D**
Model	Dell OptiPlex 7060 MFF	Micro-Star Aegis RS 12th	Intel NUC	Lenovo ThinkCentre M93P
Processor	Intel Core i7-8700	Intel Core i7-12700KF	Intel Core i3-8190	Intel Core i7-4790
Frequency	3.20 GHz	3.60 GHz	3.00 GHz	3.60 GHz
Memory	32 GiB	32 GiB	8 GiB	4 GiB
OS (Kernel)	Ubuntu 22.04.4 (6.5.0-28)	Ubuntu 22.04.4 (6.5.0-28)	Ubuntu 22.04.1 (6.8.0-58)	Ubuntu 22.04.1 (6.8.0-58)

Table 8: Detailed Device Specifications.

Name	**Brand**	**VID**	**PID**	**bcdDevice**	**Upstream**
A1	UNI	0x05e3 (Genesys Logic, Inc.)	0x0626	6.56	Type A
A2	UGREEN	0x05e3 (Genesys Logic, Inc.)	0x0626	6.56	Type A
A3	ANKER	0x291a (Anker)	0xa817	90.91	Type C
A4	TPLINK	0x0bda (Realtek Semiconductor Corp.)	0x0411	0.02	Type A
A5	ABLEWE	0x0bda (Realtek Semiconductor Corp.)	0x0411	1.01	Type A
A6	ACEELE	0x05e3 (Genesys Logic, Inc.)	0x0626	6.56	Type A
A7	OYLIAN	0x05e3 (Genesys Logic, Inc.)	0x0626	62.05	Type A
A8	AMAZON BASICS	0x2109(VIA Labs, Inc.)	0x0813	90.11	Type C
A9	Fophmo	0x0bda (Realtek Semiconductor Corp.)	0x0411	1.01	Type C
A10	ACER	0x05e3 (Genesys Logic, Inc.)	0x0626	6.63	Type C
A11	WAVLINK	0x174c (ASMedia Technology Inc.)	0x3074	0.01	Type A
A12	UtechSmart	0x05e3 (Genesys Logic, Inc.)	0x0626	6.63	Type C
B1	RSHTECH	0x2109 (VIA Labs, Inc.)	0x0822	8.b4	Type C
B2	INATECK	0x1d5c (Fresco Logic)	0x5500	1.02	Type C
B3	WAVLINK	0x2109 (VIA Labs, Inc.)	0x0822	90.14	Type C
B4	Getatek	0x05e3 (Genesys Logic, Inc.)	0x0610	94.05	Type C
C1	Dell	0x0bda (Realtek Semiconductor Corp.)	0x0409	1.55	Type C
D1	SABRENT	0x05e3 (Genesys Logic, Inc.)	0x0608	85.38	Type-A
F1	N/A	0x1d6b (Linux Foundation)	0x0003	6.05	USB Controller
SSD1	SAMSUNG	0x04e8(Samsung Electronics Co., Ltd)	0x61f5	1.00	Type C
SSD2	SAMSUNG	0x04e8(Samsung Electronics Co., Ltd)	0x4001	1.00	Type C
SSD3	GiGimundo	0x152d(JMicron Technology Corp.)	0x0584	2.12	Type C
ETH-A1	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	30.00	Integrated
ETH-A2	N/A	0x0b95 (ASIX Electronics Corp.)	0x1790	2.00	Integrated
ETH-A3	N/A	0x0b95 (ASIX Electronics Corp.)	0x1790	2.00	Integrated
ETH-A4	N/A	0x2357 (TP-Link)	0x0601	30.00	Integrated
ETH-A5	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	31.00	Integrated
ETH-A6	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	30.00	Integrated
ETH-A7	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	30.00	Integrated
ETH-A8	N/A	0x0bda(Realtek Semiconductor Corp.)	0x8153	30.00	Integrated
ETH-A9	N/A	0x0bda(Realtek Semiconductor Corp.)	0x8153	30.00	Integrated
ETH-A10	N/A	0x0bda(Realtek Semiconductor Corp.)	0x8153	31.00	Integrated
ETH-A11	N/A	0x0b95(ASIX Electronics Corp.)	0x1790	2.00	Integrated
ETH-A12	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	30.00	Integrated
ETH-B1	N/A	0x0b95 (ASIX Electronics Corp.)	0x1790	2.00	Integrated
ETH-B2	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	31.00	Integrated
ETH-B3	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8156	31.04	Integrated
ETH-B4	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	31.00	Integrated
ETH-C1	N/A	0x0bda (Realtek Semiconductor Corp.)	0x8153	33.00	Integrated
ETH-D1	TPLINK	0x0b95 (ASIX Electronics Corp.)	0x1790	2.00	Type-C
Camera1	Logitech	0x046d (Logitech, Inc.)	0x08e5	0.21	Type A
Keyboard1	MSI	0x0b20 (TransDimension, Inc.)	0db0	1.10	Type-A