Plug and Fake: On automatic forensic dataset generation of USB traces

Dennis Wolf[a,c], Lena L. Voigt[b], Harald Baier[c]

[a]*Central Office for Information Technology in the Security Sector (ZITiS), Zamdorfer Straße 88, Munich, 81677, Bavaria, Germany*
[b]*Department of Computer Science Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) Martensstraße 3, Erlangen, 91058, Bavaria, Germany*
[c]*University of the Bundeswehr Munich, RI CODE, Werner-Heisenberg-Weg 39, Munich, 85579, Bavaria, Germany*

Abstract

USB devices are widely used, versatile, and inherently trusted by the USB host, hence allowing them to subtly bypass advanced security measures. This might be a reason why USB-based attacks have become more sophisticated and numerous over the last years and target for instance highly sensitive areas, like power plants. These attacks need to be investigated, which is often done in a post-mortem analysis of the disk images. The digital forensic community agrees on the importance of digital forensic datasets (e.g., for training and education of experts) and that the dynamic change and general complexity of our IT landscape makes the automatic generation of them indispensable. Hence, in this paper our first goal is to spot the actual pervasiveness of USB-related artifacts in public datasets by reviewing the availability of synthetic Windows disk images in a semi-automatic way. Windows is chosen in this work, due to its large market share and general availability of datasets. We reveal two limitations regarding USB drive involvement in forensic scenarios: First, disk images for scenarios containing USB drives are difficult to obtain, as most scenarios do not explicitly mention the involvement of removable media – either because it is insignificant to the scenario or because its omission is intentional as part of the challenge design. Second, USB connection artifacts are unrealistic, particularly in contemporary Windows environments. For instance, among the 14 disk images in our dataset using Windows 10 or later, only five contained any traces of USB connections. Moreover, the extent of these traces was limited: only a single contemporary Windows image includes artifacts from more than two different USB storage devices. Our assumption is that lack of automation leads to such unrealistic datasets. We approach this problem by emulating USB drives on Raspberry Pi hardware, since this leaves us with the possibility to configure the USB identifiers, relevant for a forensic investigation. Furthermore, we examine the commercially available USB Rubber Ducky in this context, for it provides similar functionality. Both solutions are then integrated into a state-of-the-art data synthesis framework to enable the generation of versatile datasets, as demonstrated in two different scenarios. In our subsequent analysis of the generated data, we show that the expected traces were successfully created, making our solution a valid option for easily generating USB traces in future datasets.

Keywords: Digital forensic datasets, Automatic dataset generation, USB traces, USB spoofing

1. Introduction

Since their introduction in the 1990ies, Universal Serial Bus (USB) devices play a crucial role in campaigns aimed at delivering malicious software to both public and private computers aiming at unauthorized extraction of data [12]. Although roughly 30 years in use, exploitation using USB sticks and human engineering is still a preferred way of attackers to gain initial access to a targeted IT system [25]. Often, such social engineering attacks exploit the victim's curiosity or their altruistic intention of finding the owner of a "lost" USB device to overcome the target's sense of security and thus creating a foothold for adversaries [32].

Although the interconnection of IT systems is increasing, the threat of USB-based attacks becomes more and more popular in the past years: for instance, "[i]n the first half of 2023, Mandiant [...] has observed a threefold increase in the number of attacks using infected USB drives to steal secrets" [25]. In addition, Honeywell's GARD (Global Analysis, Research and Defense) team states in the most recent report of 2024 that increasingly more malware is focusing on infiltrating industrial systems via USB device infection [13]. Finally, the MITRE ATT&CK knowledge base describes dozens samples of *Replication Through Removable Media* as Technique T1091, where USB sticks are used as initial access vector by well-known attacker groups [4].

A key benefit of USB-based infiltration is that it does not rely on accessibility via network (e.g., from the Internet) addressing systems that are air-gapped. Air-gapping a system undoubtedly leads to a reduced attack surface, but by no means poses an insurmountable hurdle for adversaries, as for example Stuxnet has clearly shown [16]. Specialized hardware for penetration testers allows the demonstration of such physical access vulnerabilities and the testing of an organization's endpoint security defenses. In particular, the USB Rubber Ducky is a benign looking USB drive that can be programmed with a simple scripting language to inject payloads or keystrokes at extremely high speeds, al-

Email addresses: dennis.wolf@zitis.bund.de (Dennis Wolf), lena.lucia.voigt@fau.de (Lena L. Voigt), harald.baier@unibw.de (Harald Baier)

lowing for the rapid execution of sophisticated attacks on target systems. Since it utilizes the trust-by-default characteristic of the USB standard, it can bypass many software-based security measures.

The digital forensic community agrees that datasets are important for training and education of experts, tool validation, and AI model training (see e.g., [8, 11]). However, although USB sticks play an important role in cyberattacks, we show in this paper that their representation in digital forensic datasets is low. Even worse, the rapidly changing IT landscape, with its sheer multitude of applications and devices, makes a manual generation of digital forensic datasets infeasible. Thus, the automatic generation of datasets is an important area of research, our screening of available datasets reveals a use of physical USB devices for dataset generation though.

Recent advancements in data synthesis frameworks automate the execution of actions on Virtual Machines (VMs) [27, 36] and thus reduce the need for manual execution of actions on a target machine. Our concept on automation of USB trace generation hence builds on such a framework. We show how to extend a data synthesis framework to emulate/simulate USB device activity thereby eliminating the need to physically connect and disconnect a USB device during trace generation.

In all our paper makes the following contributions:

- (Semi-)automatic collection of relevant datasets: we extend the open-source Mass Disk Processor (MDP) framework [35] by three plugins to perform a metrics-related collection of datasets containing USB device artifacts (in our case Registry, Event log, and SetupAPI Log).
- USB dataset gap: our assessment of existing publicly available, scenario-based Windows disk images containing USB traces reveals a low quantity/quality of datasets and likely absence of automation during generation. The full datasheet of our collection is available online[1].
- Automation of USB trace generation: we provide an open-source USB spoofing script for Raspberry Pi hardware with integration into an automatic data synthesis framework to enable a flexible generation of individual datasets containing USB traces.
- Advanced USB attack traces: we provide the integration of a USB Rubber Ducky scenario in an automatic data synthesis framework to show the feasibility of advanced USB-based attacks.

The rest of the paper is organized as follows: Section 2 provides an overview of the USB standard including relevant forensic artifacts of USB stick usage in a Windows system and sample incidents, where USB sticks play a crucial role. Based on typical USB-specific artifacts, we are able to generate a suitable metric to verify our assumption on low quantity of USB-related forensic datasets. Thus, Section 3 reviews the current status of USB artifacts in common Windows datasets using the MDP framework, which we use to implement our defined metric. Since we can confirm our assumption about the low representation of USB devices in datasets, we present our concept and implementation to automatically spoof USB devices on a Raspberry Pi system in Section 4, using Linux kernel development tools. Our implementation has to be integrated into a suitable data synthesis framework that is selected in Section 5, followed by the implementation of two scenarios: first Section 5.1 presents an air-gapped system that is infected with a malware through a rogue USB drive. It demonstrates the ability of our solution to effectively spoof different USB devices and thus to generate more traces on a system. Section 5.2 makes use of a USB Rubber Ducky, i.e. a scenario involving specialized USB spoofing hardware. Finally, in Section 6 we evaluate our two scenarios with MDP to show the successful generation of USB device spoofing traces.

2. Background and related work

Section 2.1 starts with a short introduction of the USB standard and a general introduction of the USB enumeration process, in order to explain the subsequently discussed traces of USB connections found on a Windows machine. Section 2.2 outlines the possible means in which USB devices can be involved in cyberattacks, leading to the two scenarios we develop in this paper.

2.1. The USB standard in the context of digital forensics

The USB standard was introduced in 1996 to provide a unified connection interface between computers and various peripherals, including telephones [33]. One of the key motivations of the USB specification was *Ease of Use*. This is one reason for the lack of any authentication mechanisms and the trust-by-default characteristic of the USB standard. The USB client, the device connected to the USB host, informs the latter about its capabilities, and thus, can disguise as a different device or offer unexpected functionality, e.g., a thumb drive can act as a Human Interface Device (HID), namely a mouse or keyboard. However, it is very common for a USB device to offer multiple functions, e.g., headphones offer an audio sink and simultaneously HID capabilities to control the volume or pause the playback of music. These devices are referred to as *composite devices*. The Plug and Play (PnP) functionality and product distribution makes USB devices a perfect gateway for initial access, since drivers for common device types are usually present on the system and no additional installation is required.

Once a USB device is connected to a USB host, the USB enumeration process is started. Microsoft Windows takes the following approach [28]:

Phase I The USB host detects the presence of a device and its speed, based on the location of the pull-up resistors.

Phase II Once the host has established the connection to the USB device, it resets the device and sends a `Get_Device_Descriptor` command. The Device Descriptor contains relevant information for device identification:

[1] `https://github.com/Varg-sec/DFRWS_US_2026`

bDeviceClass, bDeviceSubClass Defines the device type, e.g., `0x09` for mass storage, and potential subclasses for more complex devices, e.g., the base class `0xE0` for wireless controllers where `0x01` refers to Bluetooth controllers.

idVendor, idProduct The Vendor ID (VID) is assigned to a company for a fee by the USB Implementers Forum. The Product ID (PID) can then be chosen by the manufacturer independently to refer to a specific device.

iManufacturer, iProduct, iSerialNumber These values point to the specific strings that provide human-readable information about a device, e.g., the name of the manufacturer or the device.

Phase III The Configuration Descriptor is read by the host to determine the number of interfaces supported by the device and its expected maximum power consumption.

Phase IV The Interface Descriptor is read by the host and provides more information about the device's interfaces and their endpoints.

Phase V Finally, the USB host needs a driver to control the USB device, which usually is done with a lookup of the VID/PID combination. The association of the driver and the device is saved in the Windows Registry, to speed up subsequent plug-ins.

The transmitted identifiers like VID and PID are usually stored in non-volatile memory within the device's controller/firmware, not in the user-accessible storage area [31]. Therefore, they cannot be changed in a trivial manner. The connection of USB devices results, depending on the Operating System (OS), in different traces that can be examined in a post-mortem analysis. Table 1 provides an overview of said traces on a Windows machine. In more recent versions of Windows, the systems ability to log and retain traces associated with USB devices has increased, resulting in a greater quantity and variety of USB-related forensic artifacts available for analysis [3, 6, 34]. Whereas the Windows Registry and the SetupAPI Log contain all traces by default, the auditing policies of the Windows Event Log need to be enabled explicitly or else it does not contain any data elements. This is also a recommended configuration by NIST [24], regardless of the OS.

2.2. USB devices in forensic incidents and scenarios

Nissim, Yahalom, and Elovici created a taxonomy of USB-based attacks [22], based on the required hardware. They identified three broad categories, including programmable microcontrollers, USB peripherals (potentially maliciously reprogrammed), and electrical attacks. Building upon these categories, they identified 29 attacks in their survey. Some of them date back to 2005, and might have lost a bit of their relevance with regard to current operating systems, e.g., Autorun exploits (ATT&CK T0895[2]). However, attacks like the one on the Ukrainian power grid in late 2022 prove that they still occur and thus are relevant for digital forensics [15].

The first category, programmable microcontrollers, offers the greatest flexibility of all the attacks, for the hardware was specifically designed or chosen to *emulate*/spoof USB devices. Each connected USB device leaves numerous traces on a computer system that might be essential for the reconstruction of events. If one wants to create digital forensic scenarios that involve the connection of multiple devices, this usually means that they have to be bought. Besides the spoofing of identifiers, some devices can also be programmed to offer unexpected functionality, like HID-capabilities. A quite popular exponent of this device branch is the USB Rubber Ducky[3]. Hak5 released the original one back in 2011 and an updated version of the attack gear in 2022. A USB Rubber Ducky looks like a normal thumb drive, but when plugged into a computer can also act as a keyboard. The device has its own scripting language, DuckyScript™, which is used to write the payloads that are placed on the microSD card of the device. In conjunction with the updated version, they introduced DuckyScript™ 3.0, which extended the command set drastically. The new USB Rubber Ducky offers composite device capabilities, e.g., still send keyboard input to the computer it is attached to, but provide the expected functionality of a thumb drive as well. It is also possible to spoof the values for VID, PID, serial number, manufacturer, and product strings. Section 5.2 demonstrates, how such a device can be integrated into a digital forensic scenario.

The second identified category, USB peripherals, concerns normal USB devices whose firmware might optionally have been modified to carry out an attack. For example, in 2020 a US hospitality provider was attacked with BadUSB sticks that mimic preloaded keyboards (ATT&CK T1200[4]), used to download the actual malware [1, 5]. This demonstrates the possibility of using USB devices as the initial stage of an attack. Moreover, academic work in this area has demonstrated the potential of using the newer USB 3.x protocol that is able to additionally capture screen content, and thus, for example, steal sensitive information or adjust the attack by using this new feedback channel [17].

The third category, electrical attacks, aims to cause irreparable damage to hardware [2] and are therefore not suitable for generating scenarios on VMs, since this type of attack would also affect the host system.

The possibilities for USB-based attacks are prolific, and their representation in digital forensic datasets does not reflect their potential. A suitable solution should generate as many of the traces discussed in Section 2.1 as possible while simultaneously being easy to use. Integration into a suitable data synthesis

[2] `https://attack.mitre.org/techniques/T0895/` (last accessed 11.06.2025)

[3] `https://shop.hak5.org/products/usb-rubber-ducky` (last accessed 05.06.2025)

[4] `https://attack.mitre.org/techniques/T1200/` (last accessed 02.09.2025)

Source	Path	Data Elements
Windows Registry	SYSTEM\CurrentControlSet\Enum\USB	VID, PID, iSerialNumber, type of device, first connection, last connection
	SYSTEM\CurrentControlSet\Enum\USBSTOR	VID, PID, iSerialNumber, volume label, last connection
	SYSTEM\MountedDevices	Volume GUID, mapped drive letter
	SOFTWARE\Microsoft\WindowsPortableDevices\Device	Volume name
	NTUSER\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist	User interaction
	NTUSER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2	User device association
SetupAPI Log	Windows\INF\setupapi.dev.log (and backup copies)	First connection
Windows Event Log	Windows\System32\winevt\Logs	Time stamps of (dis-)connection/ installation, attempt to access an object, driver install attempt

Table 1: Common digital forensic artifacts left by attached USB devices, discoverable in post-mortem analysis on a Windows machine.

framework should enable automation and thus the generation of more datasets.

3. USB artifacts in publicly available, scenario-based disk images

In this section, we assess the representation of USB connection artifacts in publicly available, scenario-based disk images of Windows systems. We compile a corresponding disk image dataset, develop an open-source implementation of USB artifact metrics, and present both the overall results and a focused analysis of contemporary Windows systems. We restrict our analysis to Windows, as it remains the most widely used desktop operating system [30] and, unlike for different Linux distributions and configurations, USB connection traces are relatively consistent across Windows versions, enabling systematic comparison [6].

3.1. Dataset Collection

To compile our dataset, in January 2026 we conducted a best-effort search for digital forensic datasets containing at least one Windows disk image with a scenario description explicitly mentioning the involvement of a USB drive. We searched Digital Corpora [7], CFReDS [29], and Datasets for Cyber Forensics [11] for the search term or tag "USB". However, the results were mainly USB-only datasets, with the only exception being the *2009 M57 Patents*[5] dataset, which includes disk images of four computers and four USB drives. Furthermore, we performed Google searches for "digital forensics dataset USB," "CFReDS USB," and "Digital Corpora USB". This way we discovered one additional dataset, namely the *2015 CFReDS Data Leakage Case*[6], which comprises a personal computer disk image, two USB drives, and one CD.

Since this search only yielded five disk images in total, we decided to expand our dataset. We hypothesize that not every scenario description explicitly mentions USB drive involvement, either because the level of detail in the scenario descriptions varies greatly or because identifying USB usage might be part of the forensic challenge.

Therefore, we also included the entire dataset indexed in [35] that comprises 25 publicly available, scenario-based Windows disk images. We extended this dataset in January 2026 by searching the aforementioned dataset collections for Windows disk images that had been published in the meantime, which led to another two disk images. Our initial search for USB-scenario disks had yielded only one Windows disk image outside this collection, as the *2009 M57 Patents* disk images were already part of it. In addition, we reviewed the scenario descriptions of the full dataset for explicit mentions of USB drive involvement. Besides the two known scenarios, only one further description contained the key word "USB". Specifically, *Ali Hadi's Challenge 2*[7] states: "You can use the image to learn the following: [...] 10. USB Forensics".

Our final dataset comprises 28 Windows disk images, six of which belong to scenarios where USB drive involvement was explicitly mentioned. The Windows versions in our dataset range from Windows XP (build 2600) to Windows 11 (build 22621). None of the dataset descriptions explicitly refer to the possible use of synthesis frameworks in image creation. However, confirming this would require a more detailed analysis of the images themselves for traces of automation frameworks.

3.2. Analysis process

After compiling our disk image dataset, we implement a set of metrics that can be indicators of USB device connections on Windows systems. For this purpose, we employ the Mass Disk Processor (MDP), which supports the collection of metrics from datasets comprising multiple disk images. MDP is a Python-based framework, that builds on the Python bindings for The Sleuth Kit, pytsk[8] and provides a plugin-based architecture for coding forensic artifact knowledge into metrics and for their automated extraction from large collections of disk images.

[5] https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario/ (last accessed: 16.01.2026)

[6] https://cfreds-archive.nist.gov/data_leakage_case/data-leakage-case.html (last accessed: 16.09.2025)

[7] https://www.ashemery.com/dfir.html#Challenge2 (last accessed: 16.09.2025)

[8] https://github.com/py4n6/pytsk (last accessed: 16.09.2025)

We choose to implement our analysis metrics within MDP rather than using standalone tools such as RegRipper[9], as MDP's Python-based, plugin-oriented design gives us precise control over the traces to consider and allows us to collect different types of forensic artifacts in one place and extend functionality as needed. We can, therefore, automatically apply the same metric-extraction workflow to the entire dataset. However, it is in general possible within MDP to run external tools and extract metrics from their results, which is why an integration of RegRipper could be implemented in the future.

For this study, we are interested in metrics related to artifacts of USB drive usage on Windows machines. We extract traces from three different locations: the SetupAPI Log, the Windows Registry, and the Windows Event Log.

Firstly, we extend an existing metric that counts the occurrences of initial hardware connections in the `setupapi.log`. While the original plugin supported Windows Vista and later, we adapt it to also process Windows XP logs. For Windows Vista and later, we additionally extract the timestamp and hardware details recorded in the corresponding log entries.

Furthermore, we add two plugins for Windows Registry metrics that use *python-registry*[10]. The first is a basic plugin that counts the number of subkeys contained in Registry locations associated with information about USB drive connections and usage, namely the `Enum\USB`, `Enum\USBSTOR`, `MountedDevices`, `WindowsPortableDevices`, `UserAssist`, and `MountedDevices` Registry keys. The second Registry plugin extracts additional details from the `Enum\USB` and `Enum\USBSTOR` Registry keys. For `Enum\USB`, it parses the key names to obtain vendor and product identifiers (VIDs and PIDs), matches them against the *usb.ids* database maintained by Stephen J. Gowdy[11], and records the key names. For `Enum\USBSTOR`, it extracts the key names directly, which already reflect resolved device information. For both keys, the plugin counts the number of subkeys per entry, corresponding to the number of unique device instances per USB type.

Lastly, we add a plugin for Windows Vista and later that uses *python-evtx*[12] to count Event Log entries potentially indicating USB drive connections. In the *System* log, we consider Event ID 20001 (driver installation attempt), counting all occurrences as well as subsets where the event description contains USB and storage or where the Setup Class corresponds to Disk Drives[13] and the device identifier starts with *USBSTOR*. In the *Security* log, we include Event IDs 4663 (object access attempt) and 6416 (new external device). It should be noted, however, that these events are only logged if the system is explicitly configured to do so, which is not the default.

[9] `https://github.com/keydet89/RegRipper3.0` (last accessed: 16.09.2025)

[10] `https://github.com/williballenthin/python-registry` (last accessed: 16.09.2025)

[11] `http://www.linux-usb.org/usb-ids.html` (last accessed: 16.09.2025)

[12] `https://github.com/williballenthin/python-evtx` (last accessed: 16.09.2025)

[13] `https://learn.microsoft.com/de-de/windows-hardware/drivers/install/system-defined-device-setup-classes-available-to-vendors` (last accessed: 16.09.2025)

Windows version	Total Number	Registry	SetupAPI Log	Event Log
Windows 11	**3**	2	2	0
Windows Server 2022	**2**	0	0	0
Windows 10	**9**	3	1	0
Windows 8.1	**3** (1)	2 (1)	1	0
Windows 7	**3** (1)	2 (1)	3 (1)	2 (1)
Windows Vista	**1** (1)	1 (1)	1 (1)	1 (1)
Windows Server 2008	**1**	0	0	0
Windows XP	**6** (3)	4 (3)	4 (3)	N/A
Total	**28**	14	12	3

Table 2: Total number of publicly available, scenario-based disk images in our dataset, grouped by Windows version (with numbers in brackets denoting disks from scenarios explicitly mentioning USB connections). "Registry, SetupAPI Log, and Event Log indicate how many images contain traces of USB drive connections in the respective locations.

3.3. Results

This section summarizes our results as depicted in Table 2, which are also available online[14]. Overall, we observe artifacts of USB connections on 14 of the 28 Windows disks in our dataset. Notably, the USB and USBSTOR Registry metrics prove the most indicative, since all disks with USB storage device artifacts contain corresponding entries in these Registry locations. The USB Registry entries also include non-storage USB devices, such as keyboards and mice. In addition, 12 of the 28 disks images contain traces of USB storage device connections in the SetupAPI Log.

The Event Log metric for driver installation attempts (Event ID 20001) implemented for Windows Vista and later, shows clear evidence of USB drive connections for only three of the 20 eligible disks, i.e., driver install attempts with the Disk Drive Setup Class and a description including the keyword "USB". While the presence of this metric is a strong indicator for USB drive involvement, its absence cannot be taken as evidence against it, since once a driver is installed, subsequent device connections may not trigger additional installation events. The same limitation applies to Security Log events 4663 (object access attempt) and 6416 (new external device) [20]. These events, although they can be indicators of USB drive connections when present, are not logged by default and therefore their absence is unsurprising even in scenarios involving USB drives. Neither of these two event types is present in any of the 28 images.

For all six disks originating from scenarios that explicitly mention USB drive involvement, we found artifacts in the USB and USBSTOR Registry. With the exception of *Ali Hadi's Challenge 2* disk (Windows 8.1), all also contain traces in the SetupAPI Log. Moreover, Event ID 20001 entries for USB storage devices are observed in both the *2009 M57 Patents* Windows Vista disk and the *2015 CFReDS Data Leakage Case* (Windows 7).

[14] `https://github.com/Varg-sec/DFRWS_US_2026/blob/main/2026-01-20_15-24-24_data_table_scenarios.tsv`

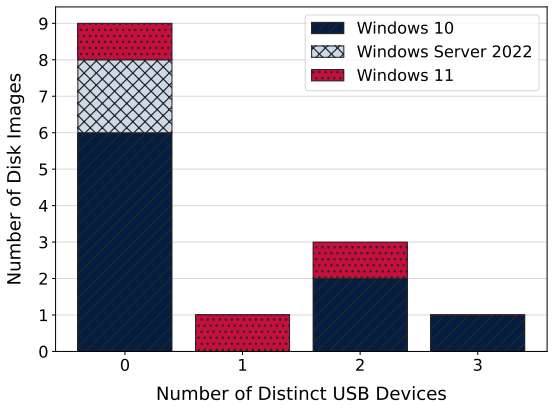

Figure 1: Number of public contemporary Windows disk images per version, grouped by the count of distinct USB device artifacts in the USBSTOR registry.

To determine the extent of USB traces in more detail, we examine the USBSTOR Registry metric for the number of distinct devices, which is based on the number of subkeys per USB type (i.e., unique device instances). The results are shown in Figure 1. We also cross-check whether additional devices appear in the respective SetupAPI Log. However, this comparison shows that some devices were listed in the USBSTOR Registry but not in the SetupAPI log, while the reverse does not occur in our dataset. Most disks, in particular ten of the 14 with any USB traces, contain only one or two distinct devices. The remaining four include two disks with three distinct USB drives, one with four, and one with six. Notably, three of these disks originate from the *2009 M57 Patents* scenario, while the last one, which contains three distinct devices, is from the *Cellebrite CTF 2021*[15].

Ultimately, to determine the representation of USB connection artifacts in contemporary Windows systems, we examine the 14 systems with Windows 10 and later (build 10240+) in our dataset, including nine Windows 10, two Windows Server 2022, and three Windows 11 disks. We use the term *contemporary* to refer to versions that are not yet at end of life as of September 2025 [19].

The proportion of disks with USB traces is lower in this subset than in our dataset overall. Only five of the 14 disks show clear indications of USB drive connections in any of our analysis metrics: one disk contains traces of a single device, three disks contain traces of two devices, and only one disk contains traces of three distinct devices, see Figure 1.

In summary, we find a limited representation of USB connection artifacts across the full dataset, especially in contemporary Windows disk images. Even in those disk images where USB traces are present, their extent, in terms of the number of distinct devices, is restricted.

[15] `https://cfreds.nist.gov/all/Cellebrite/Cellebrite2021CTF` (last accessed: 02.10.2025)

4. USB device emulation with Linux USB gadgets

This section provides our concept toward automation of USB trace generation in forensic datasets, which is based on the USB gadget subsystem.

Linux enables users the flexible userspace configuration of USB devices that are referred to as *gadgets* in this context. Two components play a relevant role for spoofing USB devices: **libcomposite** and **configfs**. The first is a kernel module that provides the infrastructure to create USB composite gadgets. With configfs[16], Linux provides a RAM-based file system that allows userspace instantiation of kernel objects. One can configure USB gadgets through configfs[17], to tell the USB host what functions the gadget provides by specifying the device's configuration. It is possible to create multiple configurations, thus multiple devices on one USB host. To enable a gadget so that the USB host can enumerate it, it has to be bound to a USB Device Controller (UDC). It is a specialized hardware component that enables a Linux system to function as a USB peripheral device. A UDC is usually not present on laptop or desktop computers, which is why additional hardware is required.

The embedded world offers a low-cost relief, since UDCs are more common here, for example in a Raspberry Pi. The Raspberry Pi Zero 2 W model is equipped with a UDC and favored, due to its versatility and cost factor. In addition, a Micro-USB cable is required to connect the Raspberry Pi to the computer, through its USB port (not the PWR IN, since it is not connected to the UDC), as shown in Figure 2.

There are already multiple projects and smaller scripts out there to configure USB gadgets on Raspberry Pis. However, most of them cannot be used for our approach, since they are too specific and for one use case only. One promising candidate is the `gadgetconfig` project of Belcarra[18]. Gadgets are configured through JSON files that are parsed with a Python program and translated into shell commands. Systemd units are used to manage configured gadgets. However, two reasons prevent the selection of this framework: First, it is not able to fully automate the creation of mass storage gadgets, since the creation of a loop device to act as a backing storage for the gadget is not automated. Secondly, the program requires a Python installation on the Raspberry Pi to generate the necessary shell scripts for the gadget management.

We provide a native shell script and omit the additional layer of abstraction, and publish it on GitLab[19]. The script automates the creation and management of USB storage gadgets, and has no further dependencies. Furthermore, it is able to pre-populate configured gadgets with files, a functionality that is elemental for digital forensic scenarios.

Some preliminary setup steps are necessary before the Pi can be used to simulate a USB device. This includes installing an

[16] `https://www.kernel.org/doc/html/latest/filesystems/configfs.html` (last accessed 11.06.2025)

[17] `https://www.kernel.org/doc/html/latest/usb/gadget_configfs.html` (last accessed: 02.09.2025)

[18] `https://github.com/Belcarra/gadgetconfig` (last accessed 26.08.2025)

[19] `https://gitlab.com/DW0lf/raspberry-pi-usb-spoofing`

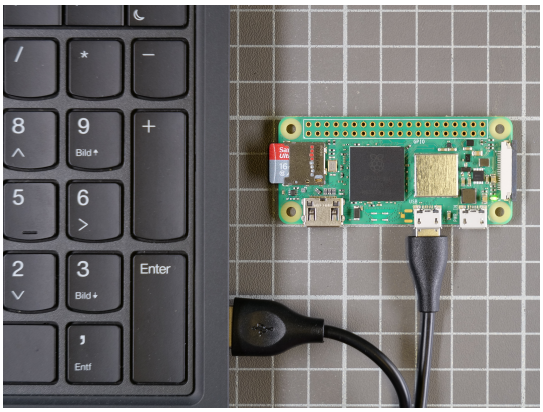

Figure 2: Setup of USB device emulation with Raspberry Pi Zero 2 W

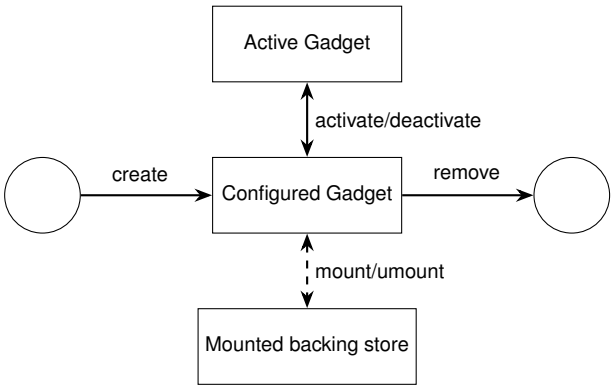

Figure 3: Life cycle of a USB gadget

OS on the system and enabling SSH access, so we can control it over the network. The README[20] provides more detailed instructions, and our script is able to test for all necessary configuration options and highlight the missing ones.

Figure 3 shows the life cycle of a USB gadget. Initially, a gadget configuration needs to be created, which can be achieved with the command shown in Listing 4.1. Each gadget is identified by a unique name, and the user may specify the device identifiers by choice. Mandatory fields have default values and are marked as such in the script.

```
# usb-spoofer create usb_1 --vendor-id=0x0951
↪  --product-id=0x160b --manufacturer="Kingston"
↪  --product="DataTraveler"
↪  --serial-number=5404A6C0AFF8F170B9600284 --size=2G
↪  --file-system=fat32 --label="MYUSB"
```

Listing 4.1: Example for USB gadget creation

Initially, a loop device is created on the Raspberry Pis file system, then partitioned and formatted to serve as the storage medium for the USB gadget. A user might want to assign a label to the created partition, to access the drive more conveniently when it is mounted. The loop device can be mounted on its own to facilitate data transfer between a computer and the device, without requiring the gadget to be configured or activated. Since the data transmission could lead to undefined behavior when the USB gadget is actively used, the script also ensures that the gadget is deactivated before the loop device is mounted on the Raspberry Pi. While the number of gadgets that can be configured simultaneously is purely limited by the total size of the file system, only one of them can be bound to the UDC to move into the active state. This is a limitation on the part of the hardware and can be resolved by providing an additional UDC.

During the next step, the gadget's configuration is created in `/sys/kernel/config/usb_gadget`. Each directory below this path represents its own gadget. The metadata supplied, such as VID and PID, is being written to the appropriate files. The structure resembles a Device Descriptor that is discussed in Section 2.1. Basically, one might follow the Linux kernel documentation to configure a gadget using configfs[21]. However, this will result in Windows showing each device's name as "Linux File-Stor Gadget USB Device", since Microsoft is using its own OS descriptors[22] that need to be added to the gadget's configuration.

All other states, shown in Figure 3, can be reached similarly, by using the gadget's name and the respective sub-command. Since the configuration of a gadget resides in memory, it is lost once the Raspberry Pi is turned off. The backing store for the loop device, on the other hand, remains on the persistent file system but can be kept optionally. This provides the possibility to reuse it for an entirely new gadget, even if it is not generated with this script.

To increase the realism of the information provided, the correct combination of VID/manufacturer name and PID/product name is recommended, as included in the previously mentioned *usb.ids* database maintained by Stephen J. Gowdy. We provide a Python script[23] that can be used to decompile this list and validate the information about extracted USB devices or generate new ones.

The question may arise whether there are any clues in the spoofed USB device that reveal it as such. Hence, we compare the outputs of the `lsusb` command (version lsusb (usbutils) 019) for a USB 3.0 and a USB 2.0 memory stick, for the following set of configurations:

1. A real USB drive on a bare metal system
2. A real USB drive inside a Kali Purple 2025.4 VM
3. The spoofed counterpart on a bare metal system
4. The spoofed counterpart inside a Kali Purple 2025.4 VM

[20] `https://gitlab.com/DW0lf/raspberry-pi-usb-spoofing/-/blob/main/README.md`

[21] `https://www.kernel.org/doc/html/latest/usb/gadget_configfs.html` (last accessed 16.08.2025)

[22] `https://learn.microsoft.com/en-us/previous-versions/gg463179(v=msdn.10)` (last accessed 16.08.2025)

[23] `https://gitlab.com/DW0lf/fortrace/-/blob/main/src/fortrace/utility/usb_spoofing/usb_device.py` (last accessed 10.09.2025)

Each case is executed on the same system, running Arch Linux with kernel version 6.18.3-zen1-1-zen and QEMU in version 10.1.2-4. First, the real and spoofed USB devices supply the same information to the bare metal system as they do to the VM, meaning, no information is changed by the virtualization or the USB redirection. Second, all information supplied by the user which is relevant for a forensic investigation is successfully spoofed. There are however two noteworthy differences: The most relevant one consists in the fact that the chosen Raspberry Pi cannot spoof USB 3.0 devices or newer ones, but is hardware-wise limited to USB 2.0. This can be observed in the output of `lsusb` and clearly identifies the spoofed device, in case it mimics a USB 3.0 one. The spoofing of USB 2.0 devices is however nearly perfect, where only values like the endpoint address are different, which does not affect the forensic analysis and is not even reported by the most common forensic tools, since these values are not used for device identification. All files that were used for this analysis can be found on the GitHub[24].

5. Sample implementation of USB infection scenarios

This section provides two sample implementations of our concept as introduced in Section 4. Due to our need for automation of dataset generation, both scenarios should be executed with a data synthesis framework. Although multiple frameworks exist, e.g., Forensig2 [21], EviPlant [26], VMPOP [23], and ForTrace [9, 10], most of them are no more available or maintained [8]. One recent representative of this kind is ForTrace++, a data synthesis framework that is under active development at the time of writing this paper [36]. The code of ForTrace++ is publicly available[25] and its modular design enables the integration of new functionality, like our USB spoofing script. It supports Windows and Linux operating systems and uses libvirt and KVM/QEMU for virtualization.

Hence, we chose for ForTrace++ and extend its functionality to pass through USB devices to the VMs, by exposing the respective libvirt functionality. A device can now be attached/detached by specifying its VID and PID. Both values are required to identify the device on the host and establish a pass through to the VM. Nevertheless, this is convenient, as these two values can be set by the command shown in Listing 4.1. Additionally, this feature enables the integration of real USB devices into scenarios, which is relevant for Section 5.2.

5.1. Scenario I: A Lonely System

The first scenario involves an air-gapped Windows 11 (version 25H2, OS build 26200.6584) system; one that is not connected to the Internet or any other public network. Often, such systems are found in the context of critical infrastructure, e.g., nuclear power plants [16]. Data transfers can happen to and from these machines by moving physical media between them,

```yaml
usb_host:
  hostname: "10.10.10.43"
  port: 22
  username: "varg"
  devices:
    - gadget_name: "usb_1"
      vendor_id: "0x0951"
      product_id: "0x160b"
      manufacturer: "Kingston"
      product_name: "DataTraveler"
      serial_number: "5404A6C0AFF8F170B9600284"
      size: 2G
      file_system: "fat32"
      label: "MYDRIVE"
    - gadget_name: "usb_2"
      vendor_id: "0x04e8"
      manufacturer: "Samsung"
[REMOVED]
    - gadget_name: "usb_3"
      vendor_id: "0x0781"
      manufacturer: "SanDisk"
[REMOVED]
```

Listing 5.1: Scenario I - USB configuration

frequently referred to as a "Sneakernet" [14]. Although this precaution introduces a high barrier to the adversary, it is not insurmountable, as several cyberattacks in the past have shown [15, 16]. After installing ForTrace++, the Windows 11 VM has to be set up, which can be automated by using the unattend-generator[26]. Although, the creation of the scenario scripts[27] takes some time, they can be reused for future executions. Recreating the scenario, including its setup, should take less than two hours.

Since the system cannot synchronize its clock with any Network Time Protocol (NTP) servers, this opens the possibility to simulate larger time spans with the connection of multiple USB devices. The functionality to manipulate the clock of a VM is already provided by ForTrace++. For this scenario, three different USB devices are configured on the Raspberry Pi. An example configuration can be seen in Listing 5.1. The remaining configuration is taken from one of the other ForTrace++ examples. More information about the configuration files can be found online[28].

ForTrace++ is being expanded by the `USBSpoofer` class, to facilitate communication with the Raspberry Pi and better integrate it into the framework's specified workflow. One can define any USB gadgets to be configured in a YAML file (see Listing 5.1), along the information about the VM to be used in the scenario. Although the USB Rubber Ducky is also able to spoof USB devices and could act as different USB thumb drives, the absence of a script-based communication channel would complicate the code of this scenario unnecessarily. Furthermore, the later conducted evaluation shows that the device does not

[24] `https://github.com/Varg-sec/DFRWS_US_2026/tree/main/USB_Device_Spoofing`

[25] `https://gitlab.com/DW0lf/fortrace` (last accessed: 16.09.2025)

[26] `https://github.com/cschneegans/unattend-generator/` (last accessed: 08.04.2026)

[27] `https://gitlab.com/DW0lf/fortrace/-/blob/main/examples/USB/Malware/usb_malware.py` (last accessed: 08.04.2026)

[28] `https://fortrace.readthedocs.io/en/latest/source/quick_start.html#yaml-configuration-files` (last accessed 05.09.2025)

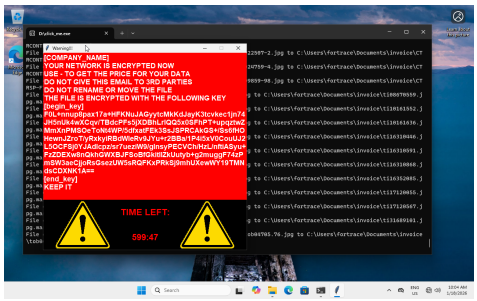

Figure 4: Windows 11 VM after infection with PoC malware

spoof the OS descriptors, hence is displayed as "ATMEL Mass Storage USB Device" in the Windows Registry.

At the beginning of the scenario, the connection to the configured Raspberry Pi is automatically established and all the specified devices are configured. Listing 5.2 shows the instantiation of a `USBSpoofer` object, which takes care of the setup and tear down automatically in the background, and thus minimizing configuration mistakes. After the automatic configuration of all specified USB gadgets, the user might transfer files to and from the associated backing stores of the devices. All methods adhere to the life cycle of gadgets, described in Figure 3.

```
with USBSpoofer(config["usb_host"]) as spoofer:
    malicious_usb =
    ↪  config["usb_host"]["devices"][1]["gadget_name"]
    spoofer.transfer_to_device(malicious_usb,
    ↪  "click_me.exe")
```

Listing 5.2: Creation of Raspberry Pi USB Spoofer and transfer of a file

A Proof of Concept (PoC) Python ransomware[29], compiled to a stand-alone executable with pyInstaller, is copied to one of the USB gadgets. Once the USB drive is passed through to the VM, a click is performed on the executable, which then starts encrypting the user directories and displaying a prominent warning message, as shown in Figure 4. This action concludes the scenario, and a disk image dump is taken for later analysis.

5.2. Scenario II: Quack back to me!

The second scenario is more sophisticated and proceeds as follows: Dave is working at E-Corp. His computer, a Windows 10 machine, is connected (as all the computers of his coworkers) to an isolated network within the company. One day, Dave's workmate Mallory leaves a benign looking USB drive on his desk that is in fact a USB Rubber Ducky. Mallory tells Dave that there is a file on the thumb drive that needs to be put in a safe spot, since he knows about the VeraCrypt container Dave is using and wants to access the data stored inside.

Several steps are required for the integration of this piece of hardware into a ForTrace++ scenario. First, it is not possible to compile the payload offline or in a script based manner, for there is neither an offline compiler nor an online API to query. Second, after transferring the payload to the USB Rubber Ducky, it has to be unplugged and reinserted to be armed. Since the device disguises itself as a normal USB stick, no script-based commands can be sent to it, as this could compromise its disguise. Thus, it is necessary to adapt the passive OS detection script[30]. Since ForTrace++ is executed on a Linux host, and this scenario involves a Windows VM, the USB Rubber Ducky blocks the execution of the payload until a Windows machine is detected. This makes it possible to execute the scenario without subsequent interaction. Similarly to **??**, ForTrace++ uses the VID and PID of the USB Rubber Ducky to forward it to the VM. This implies that the neither one of the IDs must be changed by the payload, for this would lead to its immediate disconnection. After the USB Rubber Ducky has verified its connection to the Windows machine, it starts with the execution of the ReverseDucky[31] payload. Mallory listens on a specific port for the incoming connection of the reverse shell with `netcat`. His interaction through the shell is part of the scenario script as well, thus can be adapted easily.

Mallory hosts an open SMB server in the isolated network to move files between Dave's and his machine. ForTrace++ provides *Service-VMs* for said purpose, and utilizes a Docker container[32] with a modified `smb.conf`. Mallory places a key logger executable on the SMB server and uses the reverse shell to copy it to Dave's AppData folder and subsequently execute it silently. The code of the key logger[33] has been adapted to make the creation of the log file more robust and hides them in AppData. It is published with this scenario.

In the meantime, the USB Rubber Ducky concealed the payload, while Dave proceeded to open what he assumes is a benign USB device. As Mallory told him, there is a file on the drive. Since his VeraCrypt container is locked, he unlocks it by entering his password. Finally, he moves the file to the container, and removes the USB Rubber Ducky from his computer. Meanwhile, Mallory copies the file created by the key logger to his machine, along with Dave's container, which concludes this scenario. An image of the VM is taken before it is powered down. ForTrace++ would also allow the collection of memory and network dumps. Since the reverse shell is not using any encryption but a plain TCP connection, Mallory's commands are visible in the network dumps. One is able to adjust the configuration file of this scenario to generate said dumps. However, this paper is focused on the post-mortem analysis of the disk image.

[29] `https://github.com/jimmy-ly00/Ransomware-PoC/tree/master`

[30] `https://github.com/hak5/usbrubberducky-payloads/blob/master/payloads/extensions/passive_windows_detect.txt` (last accessed 02.09.2025)

[31] `https://github.com/hak5/usbrubberducky-payloads/tree/master/payloads/library/remote_access/ReverseDucky`

[32] `https://hub.docker.com/r/dockurr/samba` (last accessed: 02.09.2025)

[33] `https://github.com/GiacomoLaw/Keylogger/blob/master/windows/klog_main.cpp` (last accessed 02.09.2025)

6. Evaluation

The MDP output for both scenarios can be found online[34]. Both scenarios are rather small and do only exist to demonstrate the easy integration of USB devices into ForTrace++ scenarios. Hence, we focus of our analysis on that part, and omit further analysis.

6.1. Evaluation of Scenario I

Upon analyzing the disk image created in Scenario I with the developed MDP plugins, we obtain results consistent with the scenario description. USB connection artifacts appear in the SetupAPI log as expected, indicating three USB storage devices from different vendors: Kingston, Samsung, and SanDisk.

As anticipated, no related entries are present in the Windows Event Log. However, the USB and USBSTOR Registry hives contain traces for only two storage device types, each with one unique device instance (Kingston and Samsung). Additionally, the USB Registry contains an entry for the USB hub and the trackpad, i.e., QEMU Tablet[35].

The last USB drive, the "SanDisk Cruzer", does not appear in the SYSTEM hive at all. Consequently, we further investigate this anomaly. We find that the behavior can be explained by the internal functioning of the Windows Registry, specifically the involvement of the Kernel Transaction Manager, which enables atomic, consistent, isolated, and durable (ACID) operations [18]. The Registry hive is not updated constantly, but pending transactions are collected in log files that are accompanying the main hives on disk, e.g., next to the `SYSTEM` hive one can find `SYSTEM.LOG1` and `SYSTEM.LOG2`. On a regular basis these log files are "flushed" which writes the transactions to the Registry. However, a log file recovery can be achieved and the pending transactions merged[36] into the main hive, to discover the previously missing USB drive. Another possibility would be to extend ForTrace++, to support image dumps of inactive VMs, since a regular shutdown of a Windows machine flushes the Registry as well[37].

6.2. Evaluation of Scenario II

In Scenario II, we again obtain results that align with the scenario description. As expected, no USB-related entries are found in the Windows Event Log. Also, the USB and USBSTOR Registry hives likewise do not contain any corresponding traces, which is consistent with the previously observed anomaly, due to the fact that a live image is obtained by ForTrace++. Again, the Registry log files would need to be merged into the main hives, to obtain the missing entries. However, USB connection artifacts are present in the SetupAPI log. In particular, it contains an entry indicating a hardware connection for a "Mass Storage" product of the vendor "ATMEL". This indicates that the USB Rubber Ducky is not able to disguise itself entirely, although the specification of VID, PID, manufacturer, product string, and the serial number, as shown in Listing 6.1, is at the beginning of the payload.

```
ATTACKMODE HID VID_05AC PID_021E MAN_Apple PROD_Keyboard
↪  SERIAL_1337
```

Listing 6.1: Attack mode specification of USB Rubber Ducky

7. Conclusion and Outlook

In this work, we evaluated the involvement of USB devices in 28 publicly available Windows datasets and have identified a significant discrepancy between the scenarios and the representation of reality.

The involvement of USB drives in digital forensic incidents, especially those concerning air-gapped systems, is likely. Current data synthesis frameworks are unable to integrate USB devices. Therefore, we have created and published a shell script that makes it easy to simulate USB storage devices on Raspberry Pi hardware. In addition, we offer integration with ForTrace++ to enable a script-driven automatic creation of such scenarios. Furthermore, we also demonstrated that our extension of ForTrace++ is able to handle penetration testing hardware as well, which enables the creation of more versatile scenarios. Since ForTrace++ is also able to execute Linux scenarios, and we showed that our solution works regardless of the OS, the provided scenario scripts can be adapted to generate Linux datasets.

While we focused mainly on the spoofing of USB storage devices, projects like *zero-hid*[38] demonstrate the versatility of the Raspberry Pi hardware, which could be used as a script-able (and cheaper) alternative to the USB Rubber Ducky. Hence, it is interesting to explore the further potential of this device or other embedded systems, which can spoof USB 3.0 hardware as well. Furthermore, Linux and macOS machines have not played a role in this work, neither in the dataset review nor in our example scenarios. A review of these datasets could yield interesting results.

References

[1] Alejandro Baca and Rodel Mendrez. *Would You Exchange Your Security for a Gift Card?* Mar. 26, 2020. URL: `https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/would-you-exchange-your-security-for-a-gift-card/` (visited on 05/23/2025).

[34] `https://github.com/Varg-sec/DFRWS_US_2026/blob/main/2026-01-20_15-24-24_data_table_scenarios.tsv`

[35] `https://the-sz.com/products/usbid/index.php?v=0x0627&p=0x0001&n=` (last accessed: 10.10.2025)

[36] `https://github.com/Silv3rHorn/4n6_misc/blob/master/registryFlush.py` (last accessed 12.09.2025)

[37] `https://learn.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-regflushkey?redirectedfrom=MSDN` (last accessed: 15.09.2025)

[38] `https://github.com/thewh1teagle/zero-hid` (last accessed: 15.09.2025)

[2] Olga Angelopoulou et al. “Killing Your Device via Your USB Port”. In: *International Symposium on Human Aspects of Information Security and Assurance*. 2019.

[3] Ayesha Arshad, Waseem Iqbal, and Haider Abbas. “USB Storage Device Forensics for Windows 10”. In: *Journal of Forensic Sciences* 63.3 (May 2018), pp. 856–867.

[4] MITRE ATT&CK Knowledge Base. *T1091: Replication Through Removable Media*. Accessed 2025-09-25. 2025. URL: https://attack.mitre.org/techniques/T1091/ (visited on 09/25/2025).

[5] Catalin Cimpanu. *Rare BadUSB attack detected in the wild against US hospitality provider*. ZDNET. Mar. 26, 2020. URL: https://www.zdnet.com/article/rare-badusb-attack-detected-in-the-wild-against-us-hospitality-provider/ (visited on 05/23/2025).

[6] Phil Froklage. *Digital Forensics: Artifact Profile - USB Devices*. Magnet Forensics. Mar. 10, 2016. URL: https://www.magnetforensics.com/blog/artifact-profile-usb-devices/ (visited on 09/08/2025).

[7] Simson Garfinkel et al. “Bringing science to digital forensics with standardized forensic corpora”. In: *Digital Investigation*. The Proceedings of the Ninth Annual DFRWS Conference 6 (Sept. 2009), S2–S11.

[8] Thomas Göbel, Harald Baier, and Frank Breitinger. “Data for Digital Forensics: Why a Discussion on How Realistic is Synthetic Data is Dispensable”. In: *Digital Threats* 4.3 (Oct. 2023). DOI: 10.1145/3609863.

[9] Thomas Göbel et al. “A Novel Approach for Generating Synthetic Datasets for Digital Forensics”. In: ed. by Gilbert Peterson and Sujeet Shenoi. Vol. 589. IFIP Advances in Information and Communication Technology. Cham, 2020, pp. 73–93.

[10] Thomas Göbel et al. “ForTrace - A holistic forensic data set synthesis framework”. In: *Forensic Science International: Digital Investigation*. Selected Papers of the Ninth Annual DFRWS Europe Conference 40 (Apr. 1, 2022), p. 301344. ISSN: 2666-2817. DOI: 10.1016/j.fsidi.2022.301344.

[11] Cinthya Grajeda, Frank Breitinger, and Ibrahim Baggili. “Availability of datasets for digital forensics And what is missing”. In: *Digital Investigation* 22 (Aug. 2017), S94–S105.

[12] Christopher Hadnagy. *Social Engineering*. Indianapolis, Ind: John Wiley &Sons, 2011. 410 pp. ISBN: 978-0-470-63953-5.

[13] “Honeywell GARD USB Threat Report 2024 | Honeywell”. In: (Feb. 2024).

[14] Andika Candra Jaya, Cutifa Safitri, and Rila Mandala. “Sneakernet: A Technological Overview and Improvement”. In: 2020 IEEE International Conference on Sustainable Engineering and Creative Computing (ICSECC). Dec. 2020, pp. 287–291.

[15] Ken Proska et al. *Sandworm Disrupts Power in Ukraine Using a Novel Attack Against Operational Technology*. Google Cloud Blog. Sept. 11, 2023. URL: https://cloud.google.com/blog/topics/threat-intelligence/sandworm-disrupts-power-ukraine-operational-technology (visited on 06/11/2025).

[16] Ralph Langner. “Stuxnet: Dissecting a Cyberwarfare Weapon”. In: *IEEE Security Privacy* 9.3 (2011), pp. 49–51. DOI: 10.1109/MSP.2011.67.

[17] Hongyi Lu et al. “BADUSB-C: Revisiting BadUSB with Type-C”. In: 2021 IEEE Security and Privacy Workshops (SPW). San Francisco, CA, USA: IEEE, May 2021, pp. 327–338. ISBN: 978-1-6654-3732-5.

[18] Mateusz Jurczyk. *Project Zero: The Windows Registry Adventure #6: Kernel-mode objects*. Project Zero. Apr. 16, 2025. URL: https://googleprojectzero.blogspot.com/2025/04/the-windows-registry-adventure-6-kernel.html (visited on 09/12/2025).

[19] Microsoft. *End of support for Windows 10, Windows 8.1, and Windows 7*. Sept. 30, 2025. URL: https://www.microsoft.com/en-us/windows/end-of-support (visited on 09/30/2025).

[20] Microsoft. *How to capture a USB event trace with Logman*. Dec. 1, 2024. URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/usbcon/how-to-capture-a-usb-event-trace (visited on 10/02/2025).

[21] Christian Moch and Felix C. Freiling. “The Forensic Image Generator Generator (Forensig2)”. In: 2009 Fifth International Conference on IT Security Incident Management and IT Forensics. Sept. 2009, pp. 78–93. DOI: 10.1109/IMF.2009.8.

[22] Nir Nissim, Ran Yahalom, and Yuval Elovici. “USB-based attacks”. In: *Computers & Security* 70 (Sept. 1, 2017), pp. 675–688.

[23] Jungheum Park. “TREDE and VMPOP: Cultivating multi-purpose datasets for digital forensics A Windows registry corpus as an example”. In: *Digital Investigation* 26 (Sept. 2018), pp. 3–18.

[24] Michael Powell, Fenimore Philip, and Stephanie Saravia. *Reducing the cyber security risks of portable storage media in OT environments*. NIST SP 1334. Gaithersburg, MD: National Institute of Standards and Technology (U.S.), Sept. 30, 2025, NIST SP 1334. DOI: 10.6028/NIST.SP.1334.

[25] Rommel Joven and Ng Choon Kiat. *The Spies Who Loved You: Infected USB Drives to Steal Secrets | Mandiant*. Google Cloud Blog. Nov. 7, 2023. URL: https://cloud.google.com/blog/topics/threat-intelligence/infected-usb-steal-secrets (visited on 05/23/2025).

[26] Mark Scanlon, Xiaoyu Du, and David Lillis. “EviPlant: An efficient digital forensic challenge creation, manipulation and distribution solution”. In: *Digital Investigation* 20 (Mar. 2017), S29–S36.

[27] Lukas Schmidt, Sebastian Kortmann, and Thomas Hupperich. “Improving trace synthesis by utilizing computer vision for user action emulation”. en. In: *Forensic Science International: Digital Investigation* 45 (July 2023), p. 301557.

[28] Shyamal Varma. *How does USB stack enumerate a device?* Oct. 13, 2018. URL: https://techcommunity.microsoft.com/blog/microsoftusbblog/how-does-usb-stack-enumerate-a-device/270685 (visited on 09/03/2025).

[29] National Institute of Standards and Technology. *Computer Forensic Reference Data Sets (CFReDS)*. Accessed 2025-09-16. 2019. URL: https://cfreds-archive.nist.gov/ (visited on 09/16/2025).

[30] StatCounter. *Desktop Operating System Market Share Worldwide*. Oct. 2, 2025. URL: https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-200901-202509 (visited on 09/2025).

[31] Texas Instruments. *VIDs, PIDs, and Firmware: Design Decisions when Using TI USB Device Controllers*. Application Note SLLA154. Texas Instruments, Nov. 2003.

[32] Matthew Tischer et al. “Users Really Do Plug in USB Drives They Find”. In: 2016 IEEE Symposium on Security and Privacy (SP). May 2016, pp. 306–319. DOI: 10.1109/SP.2016.26.

[33] *Universal Serial Bus (USB) Specification, Revision 1.0*. Available online. Compaq, Intel, Microsoft, NEC, Northern Telecom. 1996.

[34] Alexandros Vasilaras, Evangelos Dragonas, and Dimitrios Katsoulis. “USB Forensics – Recover more Volume Serial Numbers (VSNs) with the Windows 10 Partition/Diagnostic Event Log”. In: *DFIR Review* (May 26, 2021). https://dfir.pubpub.org/pub/h78di10n.

[35] Lena L. Voigt, Felix Freiling, and Christopher Hargreaves. “A metrics-based look at disk images: Insights and applications”. In: *Forensic Science International: Digital Investigation* 52 (Mar. 2025), p. 301874.

[36] Dennis Wolf, Thomas Göbel, and Harald Baier. “Hypervisor-based data synthesis: On its potential to tackle the curse of client-side agent remnants in forensic image generation”. In: *Forensic Science International: Digital Investigation* 48 (Mar. 2024), p. 301690.